Tinker: A Tiny Processor That Can

Introduction

In this project, you will implement the components of a simple processor, including the microarchitecture pipeline, fetch and decode, and a simple load/store unit. Using simulation, the concepts discussed in the class up to week 7 will come alive. A full description of the project is given in this document, and several associated documents are added to describe various parts of the architecture and help you complete and implement your design. These documents are (in the recommended order of reading):

- Project: This document. It describes the overall structure of the project, how you will be evaluated and how you will progress throughout until the end of the semester.
- Tinker Architecture Overview: A short document that describes the architecture of the Tinker processor.
- Instruction Manual: Tinker implements 32 instructions, and this document describes each of these instructions.
- Devices and Specifications: This document describes the specifications of the devices that you will implement and use to realize the microarchitecture according to your design.
- Implementing the Microarchitecture: This document describes the process by which we recommend that you go about implementing the microarchitecture.
- Instruction Fetch and Decode: A description of how the fetch and decode unit should be implemented

It is highly recommended that you read the documents first before jumping to implementation. Furthermore, a paper and pencil design may prove useful in understanding the scope of the work and scheduling the necessary tasks.

Goals

The purpose of this exercise is to tie up all the concepts that were discussed in class, including performance metrics, pipelining, multi-issue/super-scalar and out of order designs, caching, TLB and evaluation. The project also will be a good exercise in implementing a large program in the context of a team. Several components will be reused from previous homework assignments.

Overview

In this project, you will implement Tinker, a small but complete processor with many features that you will find in a real processor. The simulation will take place in a high-level language of your choice, although the use of an object-oriented programming language with inheritance is highly recommended.

You will have a reasonable degree of latitude in the design and implementation of your work. Your system will be evaluated based on the following three metrics:

- Instructions per cycle (absolute performance)
- Instructions per cycle per Watt (performance relative to energy)
- Instructions per cycle per Watt per area (performance relative to cost)

Given that the class has two teams, the winning team is the one that outscores its opponent in two out of three of these metrics. But of course, delivering a functional simulator should come first before getting too involved with the metrics.

Project Stages

Stage 1:

This stage implements a microarchitecture with a pipeline along the lines in Week 03 lecture. You will implement several devices and connect them to perform the architecture-level instructions of the processor. The implementation is described in detail in adjoining lectures. To prevent "throwing hardware on the problem", area and wattage are tracked so that your solution will be judged based on the resources you utilize. There is no free lunch.

Stage 2:

In this stage, you will reuse the simulation work from Homework 04 to connect to the microarchitecture that you implemented in stage 1. This will complete the simulation of the processor for the purpose of the exercise.

Stage 3:

Performance evaluation.

Software Engineering and Logistics

The project is considerably involved. Tenets of software engineering such as modularity in design, defensive programming, regression testing, and a project plan will put some order and reduce stress. It is recommended that you put a project plan with deadlines so that you can scope the effort well. Adhoc and brute force work will not likely yield satisfactory result and may cause undue stress and anxiety. Starting early is *imperative.*

Several test cases will be released to help you in the development and debugging. Also, you should write your own test programs if possible. Other tools will be provided to help in the project in general.

Effort Distribution

The project is quite involved, and this requires every member of the team to really carry their assigned weight. This is not a project where one hero programmer can carry the day. The task is not complex, but its size is substantial. It is designed for four students working together as a team. Therefore, it is important that good teamwork habits be adhered. Stay positive and communicate promptly and frequently. Make sure you define the interfaces between the various parts of the code.

Deliverables:

Stage 1 is due November 24th.
Stage 2 is due December 1st.
Stage 3 and the entire project is due December 13th.

Magic

Ideally, a project of this kind should be implemented in a logic design environment (e.g., using VHDL or Verilog). But this would be too slow for the purpose of this exercise. There will be a strong temptation to take short cuts and "implement things in C++". To the extent that is possible, please try to mimic how logic operates. Complex decisions based on elaborate and complex functions in C++ are not likely to emulate reality. Remember that hardware is good at testing logic, moving data, indexing, and performing simple and primitive operations. It is not usually possible to just inspect the instructions, for instance, and come up with a complex way of identifying dependencies. Therefore, we would like to be faithful to the task of implementing the work with as much fidelity as possible.

Getting Help

As has been the policy in this course, you are free to borrow, reuse, copy, or rely on external sources as much as you desire or need. But please document any resources that you use in this manner. You may not find it possible to reuse much code out there, given the peculiar nature of this project. But then, there may well be very useful components out there that you can adopt and adapt. When in doubt about anything, please do not hesitate to call on the instructor.

Happy coding, simulating, and evaluating.