[Lab07] Cache

Junhyuk Choi, Bogyeong Park, and Yujin Chung

arch23-ta@hpcs.snu.ac.kr

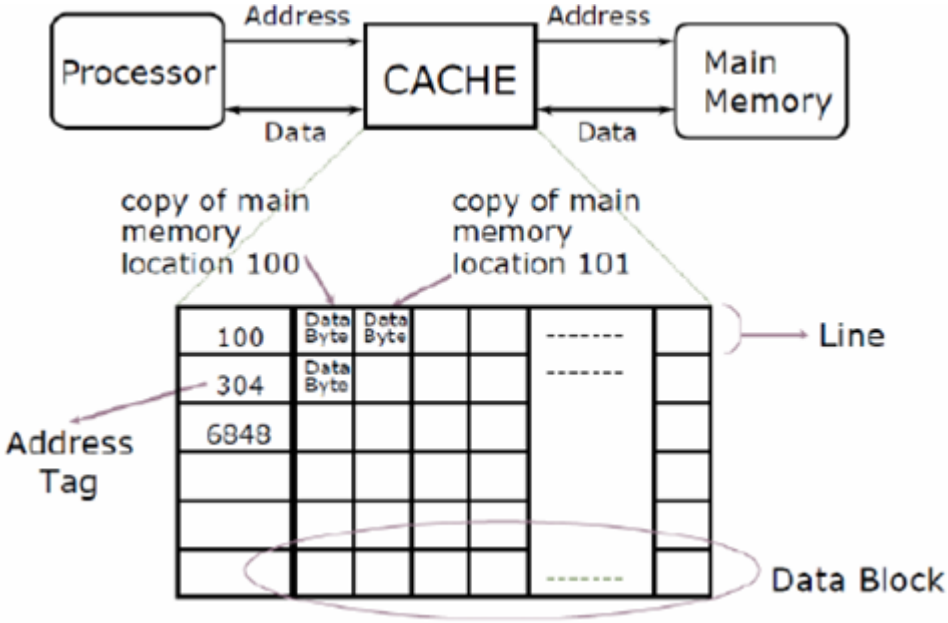
Overview

In this assignment, we are going to implement a cache on our previous CPU design, a pipelined CPU. We also evaluate the speedup achieved by using cache by comparing no cache version CPU with our new CPU.

Goal

- Understand what a cache is.
- Understand why cache improves the performance of a CPU.
- Design and implement a direct-mapped cache on your pipelined CPU.

Background



The above figure shows the structure of a typical cache system. A cache mitigates the performance gap between CPU (a few cycles) and memory (a few hundred cycles). Compared to memory, a cache has a small capacity, but it can achieve a huge performance improvement because of data locality.

Note

From this assignment, we will use the previous CPU design (pipelined CPU) to implement the cache.

Assignment Description

1. Implement the cache in the pipelined CPU which supports every TSC instruction. Please refer to "**[Lab05] tsc_ISA.pdf**"
2. Your CPU should pass all test cases we provided.
3. Evaluate the benefits of using cache (compare a baseline CPU and a new CPU)
4. Elaborate which policies you chose & how you modeled memory latency for various cases with waveforms

You should change your memory module (memory.v) to satisfy new latency models (described in a lab material)

Please refer to the lab material "**[Lab07] Cache.pdf**" for more details.

Note) you should submit both designs: the baseline CPU (satisfying the new latency model) and the new CPU (w/ cache).

Grading

Design

We will evaluate whether your CPU module followed the two design constraints as follows:

- Proper latency model for both baseline and new CPUs.
- Proper cache requirements.

Functionality

We will grade your CPU module based on the pass/fail result of the provided testbench (see "cpu_tb.v"). The testbench contains most of the pass/fail tests that we will check, but we may add a few other pass/fail tests in grading.

Code review

Your code should be easy to read. If not, you may get some penalties.

- You should provide explanations of the module's inputs, outputs, and functionality in the comments.
- All the variables, parameters, and file names should be self-explanatory. It should be neither "reg a", "wire wire1" nor "helloWorld.v".
- You should stick to a consistent code style (e.g., indentation, naming).
- Do not mix blocking and non-blocking assignments in a single always block.
- Do not use delay (e.g., #10) to correct the timing. Delays are only for testbench.

Report

Basically, please refer to “How_to_write_your_report.pdf”.

We'll follow the previous guideline for this week

You **should explain** how you implemented a cache, calculate the **hit/miss ratio**, and **compare the performance** with the baseline CPU.

You should elaborate which **policies** you chose & how you modeled **memory latency** for various cases with **waveforms**.

Submission

Due: 05/22 (Mon.) 19:00

- Late submission
    - 10% penalty per day until 05/27 (Sat.) 19:00
    - NO points after 05/27 (Sat.) 19:00

Upload a single zip file named LabXX_groupXX_20XX-XXXXX including your (1) project folder, (2) report on the eTL board.

- For example, if your student number is 2023-12345, and your group number is 67, submit Lab07_group67_2023-12345.zip
- Check “copy sources into project” option in “add or create design sources”.
- **Make sure your Vivado project contains your source code.**
    - Check the Verilog files inside “[vivado_project_name].srcs/sources_1/new/” before submission.
- Do not delete files in the project folder if unnecessary. (e.g., .xpr file)