ADS I Class Project
Hardware Design with Chisel RPTU Kaiserslautern-Landau
Prof. Dr.-Ing. Wolfgang Kunz Fachbereich Elektrotechnik und Informationstechnik
M.Sc. Tobias Jauch Entwurf informationstechnischer Systeme

Project Task: Branch Target Buffer

The final assignment in our ADS I Class Project is to design an advanced hardware module according to the specification provided below. Your task is to implement a Branch Target Buffer, as described in the ADS I lecture (slide 6-48), along with a dynamic 2-bit branch predictor (slide 6-47).
A Branch Target Buffer (BTB) is a hardware component used in modern processors to improve the efficiency of branch instructions. The BTB acts as a lookup table for target addresses. When the control flow encounters a branch instruction, the BTB checks if it contains an entry for this branch and what outcome the prediction scheme used by the BTB predicts. This allows the processor to quickly fetch the next instruction without waiting for a full evaluation of the branch. If the prediction is correct, the processor continues execution without delay. If it is wrong, the processor rolls back and restarts execution from the last committed instruction. In both cases, the prediction is updated to reflect the current behaviour of the program. This ways, a BTB helps to reduce pipeline stalls and improves overall performance.
The BTB serves as a cache for target addresses. Therefore, we can utilize strategies known from data caches, such as associativity (cf. slide 7-23), to make the BTB more efficient.

Structure of the Project Task

This last project task does not provide any Chisel project or structure yet, but after the previous assignments, you surely can manage to start a Chisel design from scratch.

Specification

The goal of this project is to design and implement a 2-way set-associative Branch Target Buffer (BTB) with 8 sets using CHISEL. Your design should operate as an independent module, interfacing with the rest of the processor through a predefined interface.

The BTB consists of 8 sets, each containing 2 ways, making it a 2-way set-associative structure. Each entry in the BTB should include:

- A **valid bit** to indicate whether the entry contains usable data.
- A **tag** to identify the branch instruction associated with the entry.
- A **branch target address**, which provides the predicted next program counter (PC) when the branch is taken.
- A **2-bit predictor state** indicate whether the branch is predicted to be taken or not.

Each entry in the BTB contains a 2-bit saturating counter (similar to the one shown on slide 6-47 in the ADS I script) to predict whether the branch is likely to be taken or not taken. In a 2-bit counter, a prediction must be wrong twice before it is changed. This enhancement allows the BTB not only to predict the target address of a branch but also to predict whether the branch will be taken or not. The branch prediction mechanism implements a 2-bit finite state machine (FSM) with four states:

- strongTaken
- weakTaken
- weakNotTaken
- strongNotTaken

The BTB uses bits [4:2] of the input program counter to index into one of the 8 sets. For each set, the two ways should be checked for a tag match. If a matching tag is found and the corresponding entry is valid, the BTB should output the predicted branch target address. If no valid match is found, the BTB should indicate that no prediction is available (*valid := false.B*). In this case, a new entry will be written after the branch target has been calculated in EX stage (*update* signals). Do not forget to also initialize the 2-bit predictor FSM for a new entry.
Since each set has two ways, the design must implement a Least Recently Used (LRU) replacement policy. This policy ensures that when a new entry is added to a full set, the entry that has not been used for the longest time will be evicted.
The BTB module uses the following I/O signals:

Inputs:

- **PC:** A 32-bit program counter representing the address of the branch instruction being fetched or executed.
- **update**: A 1-bit signal indicating whether the BTB should be updated with new information.
- **updatePC:** A 32-bit program counter associated with the branch instruction being updated.
- **updateTarget:** A 32-bit branch target address to be stored in the BTB.
- **mispredicted:** A 1-bit signal indicating whether a branch was actually taken during execution (used to update the predictor).

Outputs:

- **valid:** A 1-bit signal indicating whether the BTB has a valid prediction for the provided program counter.
- **target:** A 32-bit signal representing the predicted branch target address when a valid prediction exists.
- **predictedTaken:** A 1-bit signal indicating whether the branch is predicted to be taken or not.

When the update signal is asserted, the BTB should update the appropriate set with the updatePC and updateTarget values. In addition to updating the target address and tag, the predictor's state must also be initialized. If both ways in the set are already in use, the LRU replacement policy should be used to determine which entry to evict.

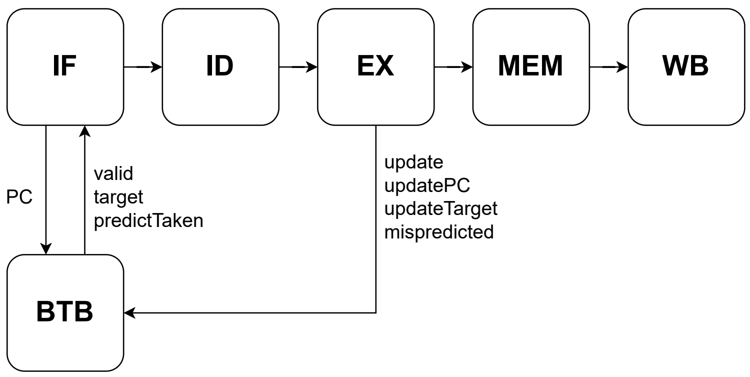

Figure 1: I/O Signals

An update always occurs two clock cycles after an entry has been read, because the calculation of the branch target is done in the execute phase. Therefore, it is important to ensure that the update mechanism still selects the correct entry when a new target is written and the 2-bit predictor FSM is updated.

Preparation

Before you start with the implementation, please answer the following questions:

1. How does a higher associativity affect the performance of a cache?
2. What's the best initial state for the 2-bit predictor FSM? What effects would different initial states have in regular program patterns (e.g., loops)?

It might also help you to think about the following details before starting with your implementation.

1. Find a way to implement a structure of registers with 8 sets and 2 ways. Standard memory classes might not be a good option here.
2. How can the number of index bits be determined?
3. How many bits does the tag of each BTB entry need in a configuration with 8 sets and 2 ways?

Implementation

Implement a 2-way set-associative BTB according to the specification above in your own Chisel project.

Test your Implementation

Create test cases to verify your BTB's functionality. These tests should demonstrate the following:

- Correct predictions for program counters with valid entries.
- Handling updates to the BTB properly.
- Correct eviction behavior based on the LRU replacement policy.
- Demonstrate accurate state transitions of the FSM and correct predictions based on the current state.