Trevor Morris and Tristan Seroff
5/19/16

ECE 154B Project 3 Report

Introduction

Here we describe how we will implement branch predictors for our MIPS processor.

Design Methodology

Step 1) Static Prediction

Our design currently already performs a static “always not taken” prediction. The next instruction after each branch is always fetched into the pipeline. The branch condition is checked at the decode stage, and if the branch is taken the fetch stage is flushed. Otherwise the next instructions run as normal.

How does this improve the performance? Branch prediction misses cost a total of 2 cycles for the flushed instruction. Branch prediction hits(not taken) cost 1 cycle for the branch instruction.

How do we deal with EX, MEM and WB registers when prediction is wrong?
The mispredicted instruction is flushed before it even reaches the decode stage. EX, MEM, and WB registers are unaffected. The instruction simply becomes a ‘nop’ and does nothing as it passed through the pipeline.

Step 2) Dynamic Prediction

To perform dynamic branch prediction, during the instruction fetch stage the branch target buffer is searched for any entries which match the lower bits of the current program counter. If an entry is found for that branch PC, the prediction state bits are checked to see if the branch is predicted to be taken or not taken.

If the branch is predicted to be taken, the predicted PC from the BTB is used.

If the branch is predicted not taken, or not present in the BTB, PC increments normally.

The determined next PC is fed back to the fetch stage for the next clock cycle, while the branch instruction advances to the decode stage. The result of the instruction is computed.

Non-branch instructions don’t cause any action to be taken.

If the branch wasn’t present in BTB and is taken, it is added and next instruction flushed.

If the branch was predicted taken but isn’t, BTB is updated and next instruction flushed.

If the branch was predicted as taken and is resolved as taken, no action is needed.

New Hardware

Branch Target Buffer Module:

Contains 2^7 entries, each one having a 32-bit Branch PC, Predicted PC, and a 2-bit prediction state.

Has an input ‘PC’. Lowest 7 bits of PC are used to index the entry table, and a comparison is done between the Branch PC and the input PC. The result of the comparison is output to ‘entry_found’. The Predicted PC at the index is output to ‘predicted_pc’.
Has an input ‘branch_address_in’ and ‘predicted_address_in’, and a flag ‘btb_write’. If write is 1, the new entry is entered into the table, with the state set to weakly taken.
Has an input ‘state_change’ and ‘state_write’. If ‘state_write’ =1, the state of the predictor for ‘branch_address_in’ is changed according to ‘state_change’(0 is N and 1 is Y).

Modifications to Pipeline

IF Stage:

PC is given to the BTB, which checks if the address exists in the table. If it does not, PC+4 is put as next PC. Otherwise the next PC is set to the predicted pc from the BTB. This is done with a multiplexer.
‘entry_found’ is buffered to the next stage.

ID Stage:

Currently comparators determine a branch is to be taken in the decode stage and pass the information to the decode buffer to flush it if needed. We would instead take the result of the comparator(‘taken_branch’) and pass it into the execute stage buffer.
‘entry_found’ is buffered to the next stage.
‘taken_branch’ is generated if a branch instruction is decoded with a positive result.
‘Pc’ is buffered to the next stage.
‘PCBranchD’ from the is buffered to the next stage.

EX Stage:

If ‘entry_found’ is 0:

If ‘taken_branch’ is 0: execute instruction normally.

Else If ‘taken_branch’ is 1: Enter branch instruction address(PC) in ‘branch_address_in’ and next PC(PCBranchE) into ‘predicted_address_in’ and set ‘btb_write’ to 1. Kill fetched instruction. Restart fetch at the correct target.

Else if ‘entry_found’ is 1:

If ‘taken_branch’ is 0: Mispredicted branch, kill fetched instruction; restart fetch at the correct target. ‘State_write’ = 1, ‘state_change’ = 0, branch_address_in=PC

Else If ‘taken_branch’ is 1: Branch correctly predicted; continue execution with no stalls. ‘State_write’ = 1, ‘state_change’ = 1, ‘branch_address_in’=PC

The ‘entry_found’ ‘taken_branch’, ‘Pc’, and ‘PDBranchD’ signals are buffered through the execute stage buffer and fed back to the Branch Target Buffer, which updates its prediction values based on the result. A taken branch increments the internal prediction bits, a not taken branch decrements them. If the branch was taken but was not present in the Branch Target Buffer, it is added as a new entry.

Hazard Unit:

Connects to ‘entry_found’ and ‘taken_branch’ from the decode stage, and changes pipeline control on the next clock cycle if necessary, corresponding to the branch instruction advancing to the execute stage. If the branch was mispredicted, the hazard unit flushes the instruction that was fetched from the incorrect PC.

Step 3) Two Level Correlating Global Prediction

Modify the BTB to have 4 entry tables. Create a 2-bit global history register which acts as a FIFO queue; Every time a branch instruction is decoded, its result is fed into the global history register and the oldest history value is pushed out. The global history register indexes the entry table to use in the BTB.

The specific entry table in the BTB chosen from the global history register is read to make a branch prediction and updated once the branch result is computed in the same manner as the one-level branch predictor in Step 2. The index of the entry table used to make the branch prediction is buffered through the pipeline alongside the branch instruction itself, so that it can be updated once the result is computed.