HW#2 Branch Predictor Design

Chun-Jen Tsai
NYCU
10/13/2023

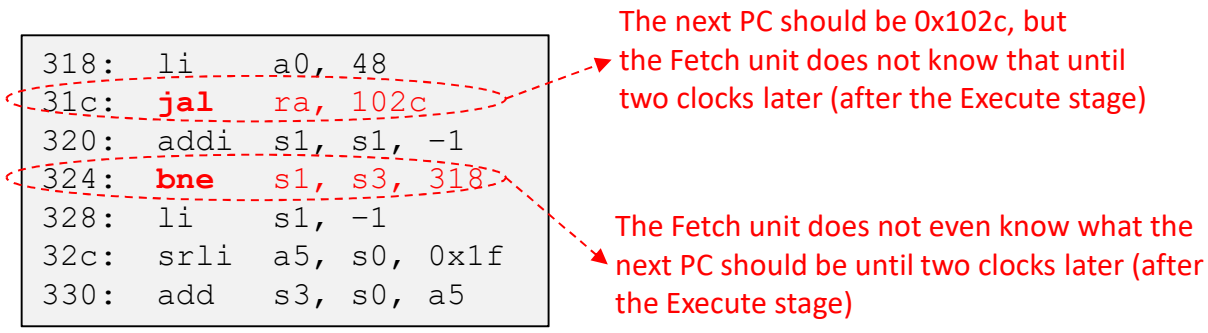
Homework Goal

- This homework requires you to modify the Branch Prediction Unit (BPU) of Aquila to improve CoreMark
  - Analyze the current BPU first
  - Design a two-level predictor to improve the performance
- You have 2+1 weeks to implement this homework
  - You must upload your initial analysis report by 10/27, 17:00.
  - Students who have finished the analysis report get one extra week for preparing the final report. You should upload the final report and your code by 11/3, 17:00.

Types of Branches

- There are three types of branches
  - Conditional forward jumps: for if-then-else statements
  - Conditional backward jumps: used in looping statements
  - Unconditional jumps: for function calls, or from bad coding

- The problem of branches:



Dependencies of Fetch on Execute

- In a 5-stage pipeline, a branch instruction, after Fetch, may take up to two cycles to determined the next PC
  - The Execute is responsible for calculating the branch condition and update the PC
  - Do we have to stall the Fetch stage for the next instruction by two cycles?
- A branch predictor predicts the PC before Execute computes the condition expression
  - If the prediction is wrong, the pipeline has to be flushed before the Memory stage!

Static Branch Prediction

- Static branch prediction always make the same decision (forward/backward × taken/not taken)
- Implementation can be done by one of three methods
  - Hardwired into the processor pipeline
    - Assuming branch always taken, the Fetch must do a quick decode of the target PC
    - Assuming branch always not taken, then the PC ← PC + 4
  - Compilers generate the hint bit if the ISA supports it
  - Cooperation between the processor and the compiler, by following some register usage convention. For example,
    - "`bne s1, s3, 318`" suggests taken
    - "`bne s1, s4, 318`" suggests not taken

Dynamic Branch Prediction

- The processor collects statistics at runtime of whether every branch instructions are taken or not
- The fetch unit fetches the predicted next instruction
- In the case of a misprediction, the pipeline has to be flushed to re-fetch the correct instruction
  - The penalty is high for a misprediction
  - The CPU states has not been changed upon misprediction

Stage / #Cycles	Fetch	Decode	Execute	Memory	Writeback
1	BR	?	?	?	?
2	ADDI	BR	?	?	?
3	SLL	ADDI	BR	?	?
4	?	NOP	NOP	NOP	?

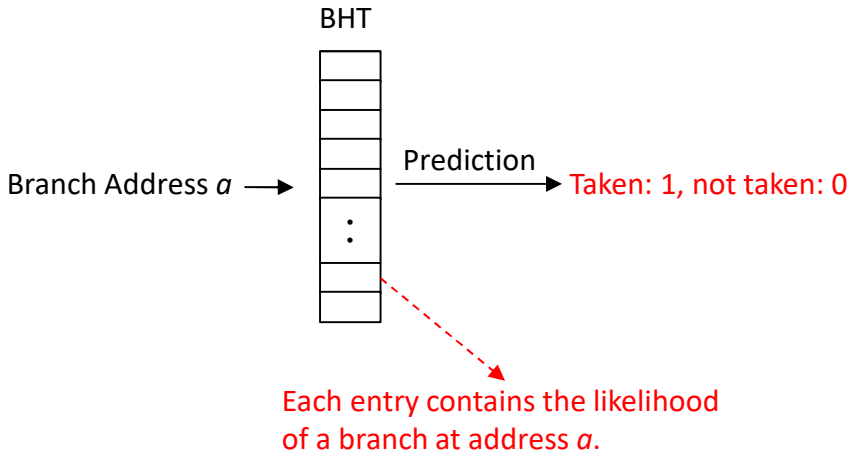
Next PC determined!

Branch Prediction Schemes

- One-level Predictor
  - Uses a Branch History Table (BHT) indexed by the recent branch addresses
  - When the fetch unit reaches a branch location, it gets the PC for the next instruction to fetch based on the BHT
- Two-level Adaptive Branch Prediction
  - MCFarling's Two-Level Prediction (gshare, 1993).
  - Call-stack predictor for function call returns

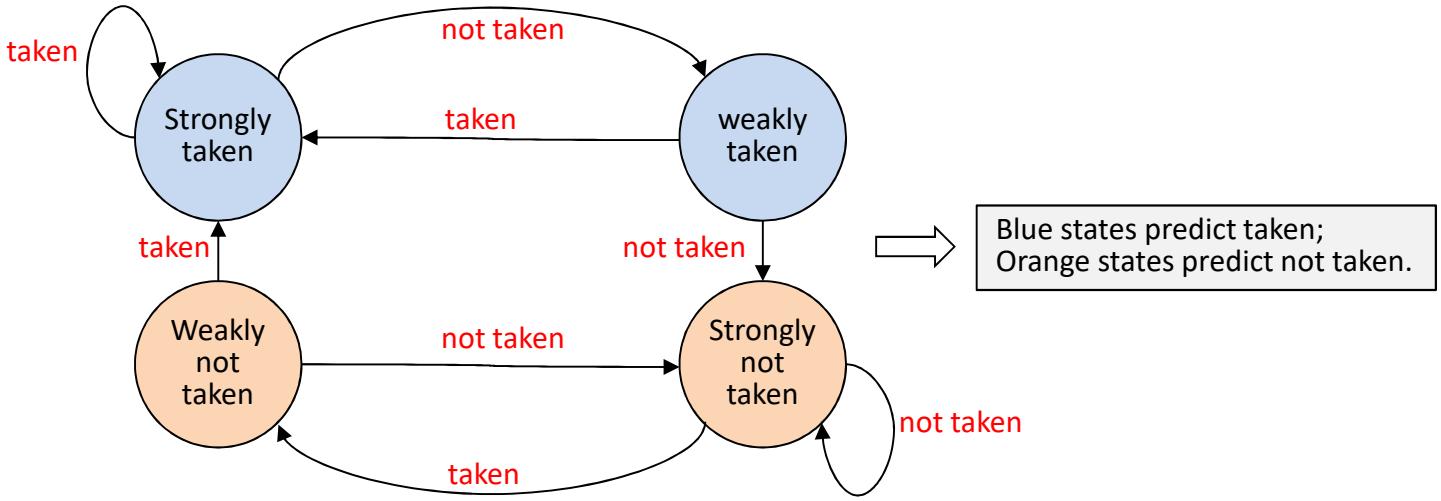
One-level Branch Predictor (1/2)

- One-level predictor is based on bimodal assumption:
  - The branch at a specific PC is either taken or non-taken most of the time
- For one-level branch predictor, we must determine:
  - How many address bits are used to index the BHT
  - How many bits are used to record the branch statistics
  - How many branch instructions are recorded in the BHT



One-level Branch Predictor (2/2)

- Aquila implements the simple 2-bit predictor
  - For each branch instruction, we record its branch likelihood with one of four possible states:

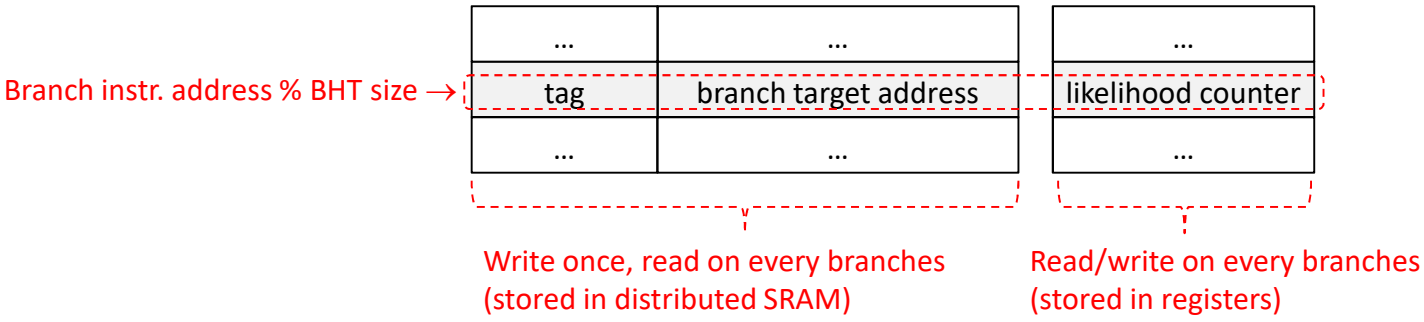


  - The state changes after the execute stage determines whether the branch is taken or not.

BHT Implementation

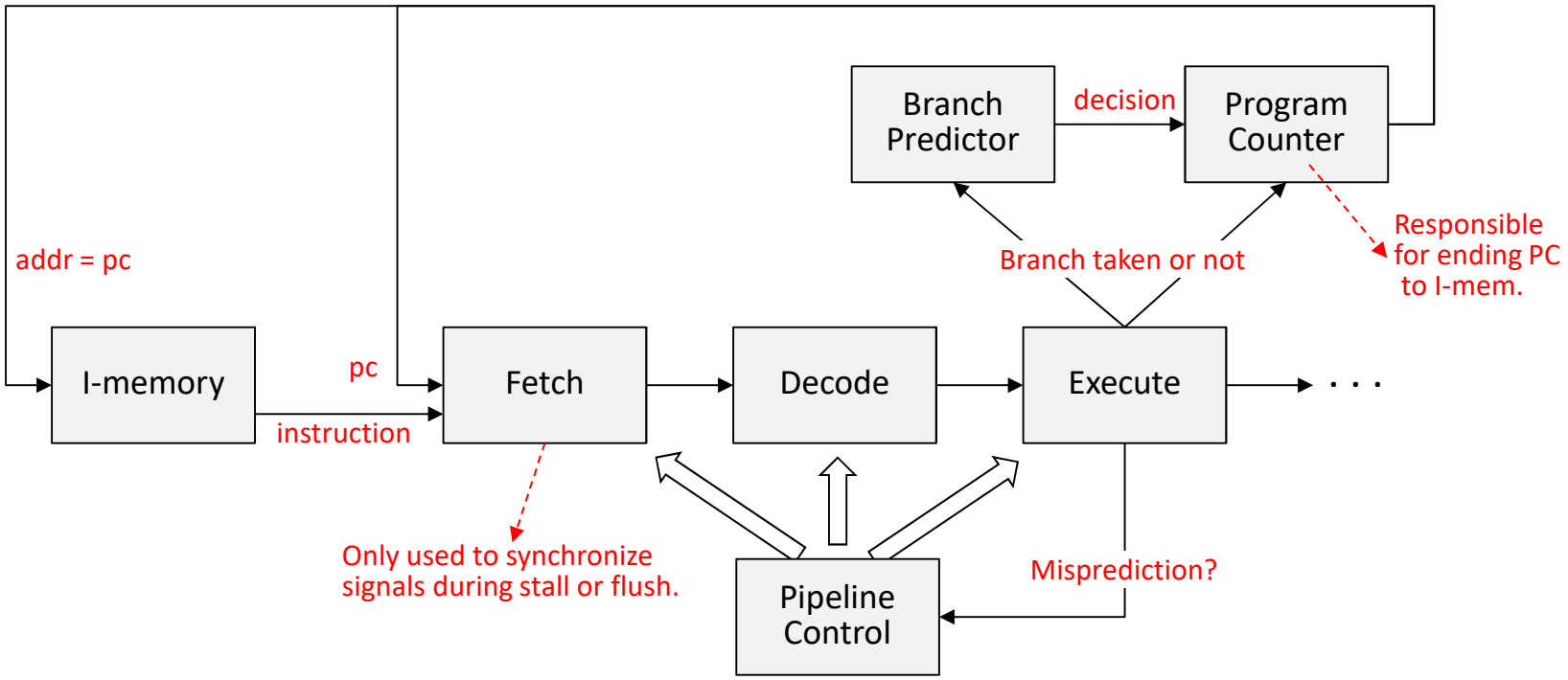
- The BHT is a tagged memory similar to caches
  - Typically, direct-mapping by the branch instruction address
    - The least significant bits is used as the index to the BHT (e.g. 5 bits for 32 table entries)
    - The full address is used as the tag to determine hit or miss
  - As an alternative, expensive fully-associative BHT can be used for better performance at higher circuit cost

- BHT table entry format:

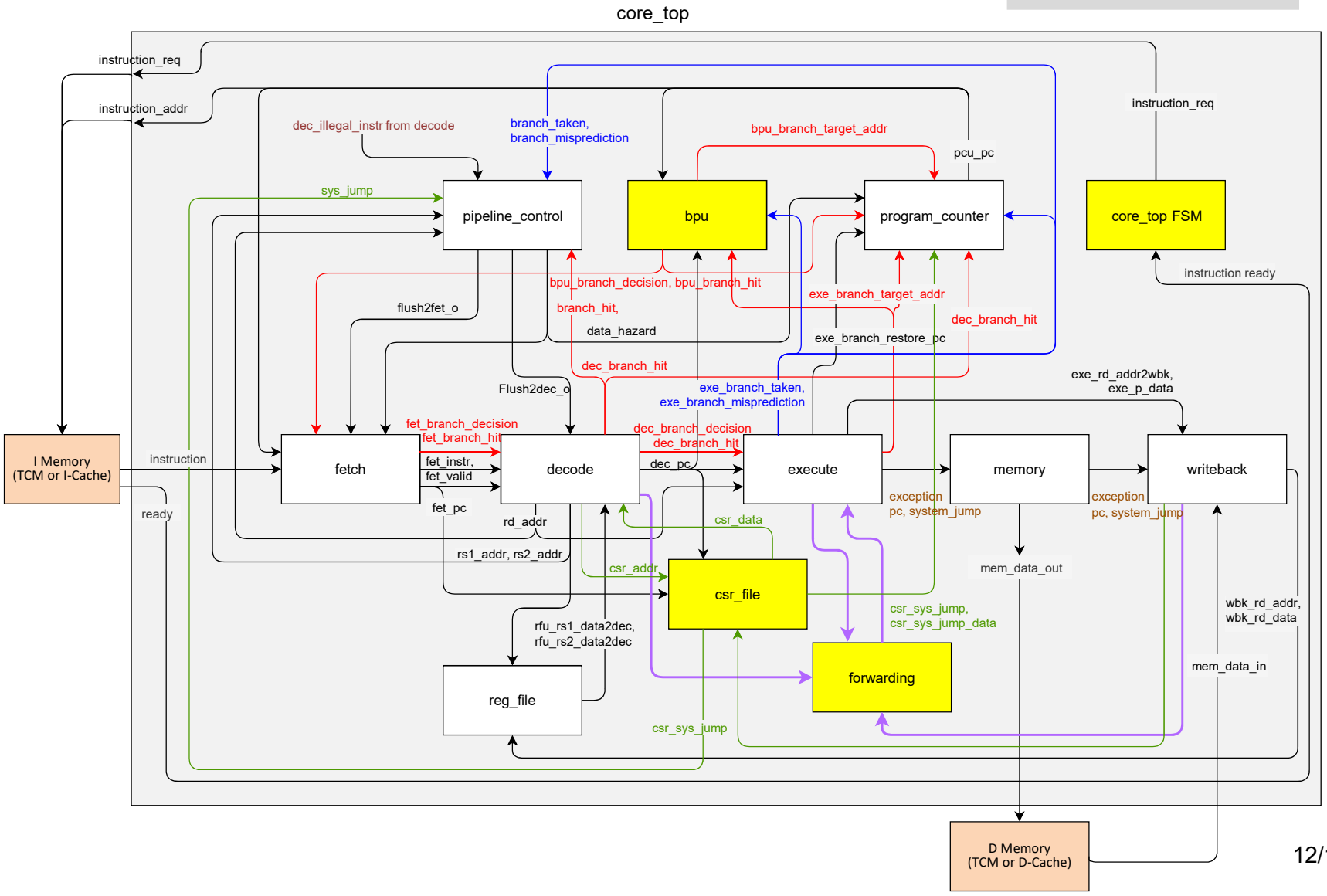


Branch Prediction Flow in Aquila

- A branch predictor tells the fetch unit which instruction to fetch before the branch has been executed
  - Aquila only predicts the target for branches and `jal`, not `jalr`

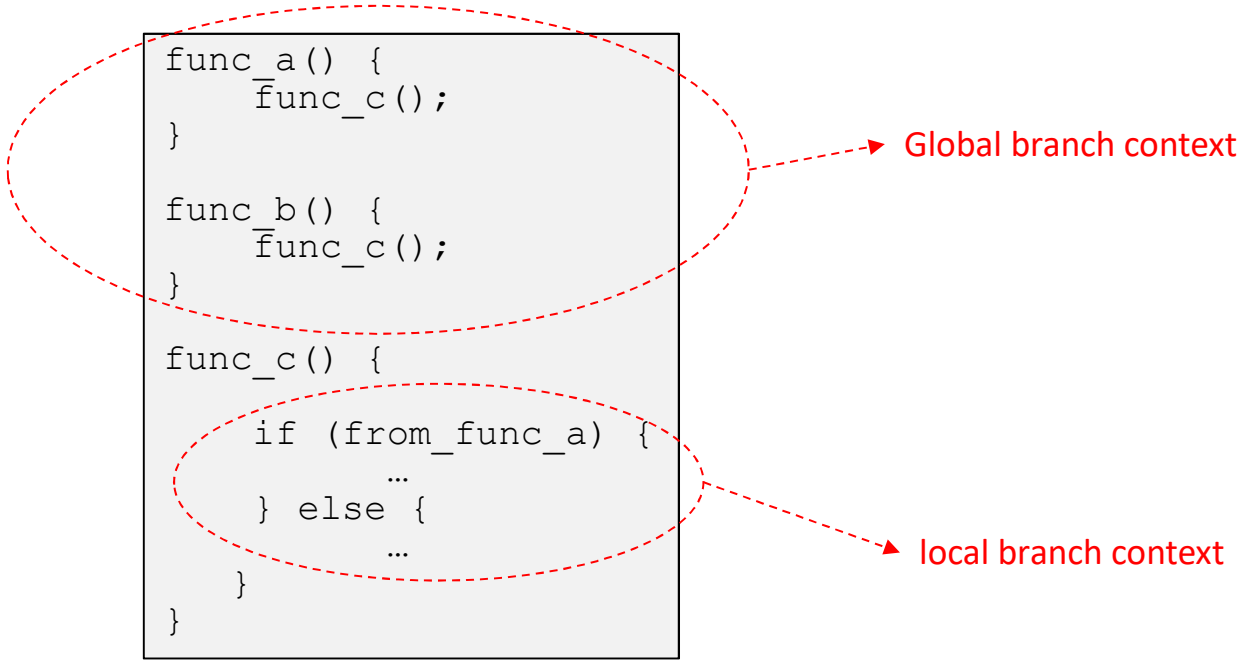


Aquila BPU Related Signals



Basic Ideas on Two-Level BPU

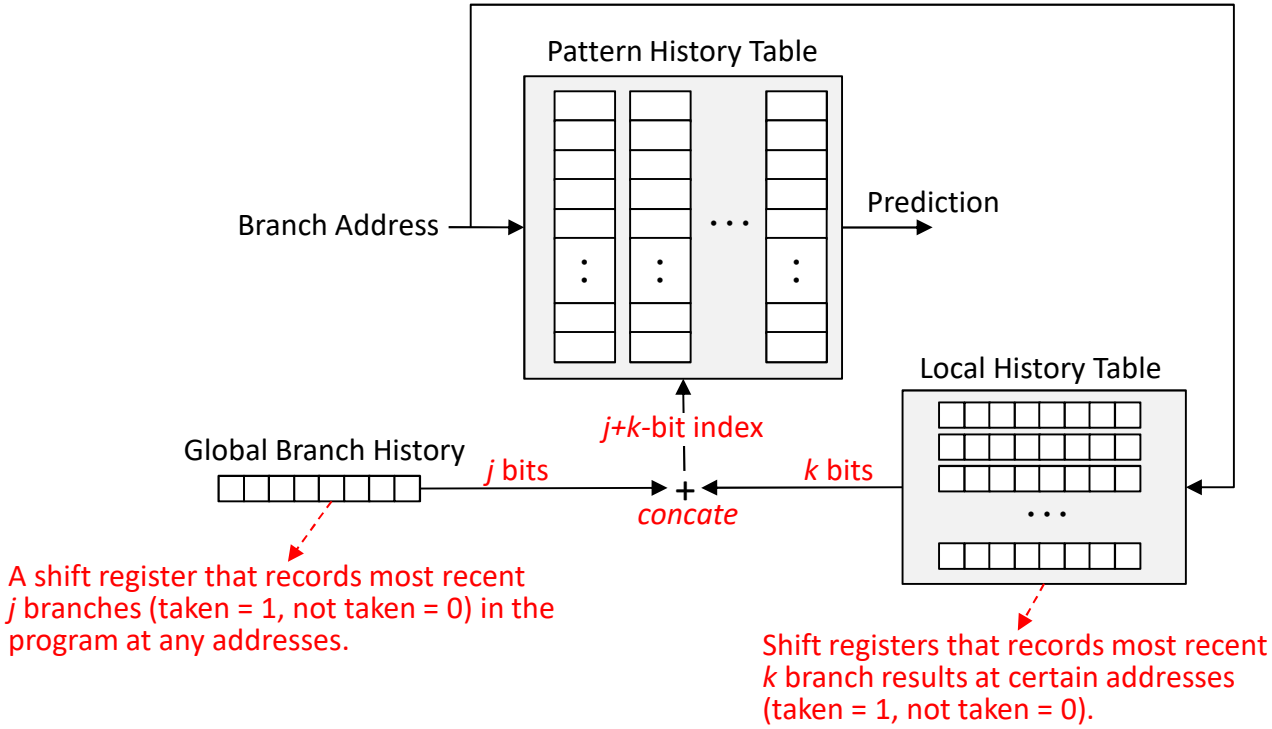
- One-level predictors lack global context information



- 2-level predictors consider both global & local contexts

Two-level Branch Predictor (1/3)

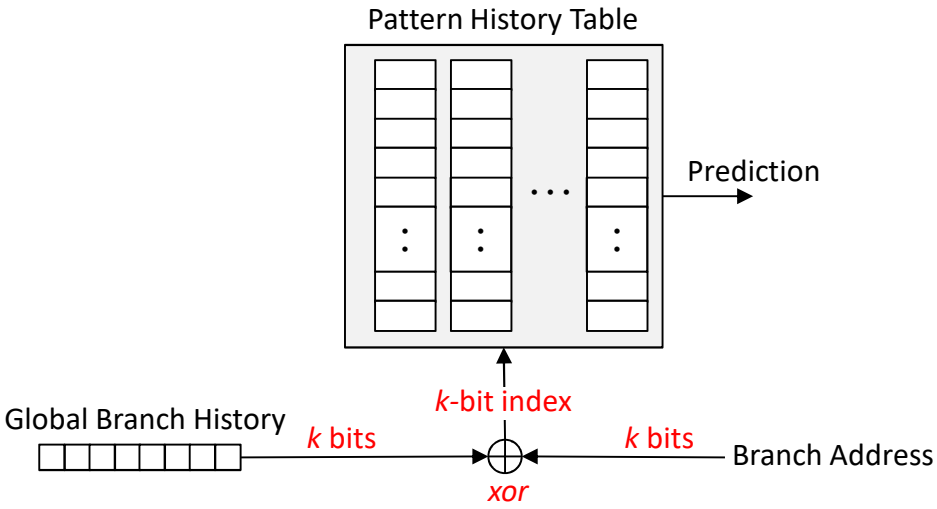
- A branch depends on the branch address as well as:
  - Nearby branches – recorded using Global Branch History
  - Longer history of the same branch – Local Branch History



S. McFarling, "Combining Branch Predictors," WRL Technical Note TN-36, Digital Equipment Corporation, June 1993.

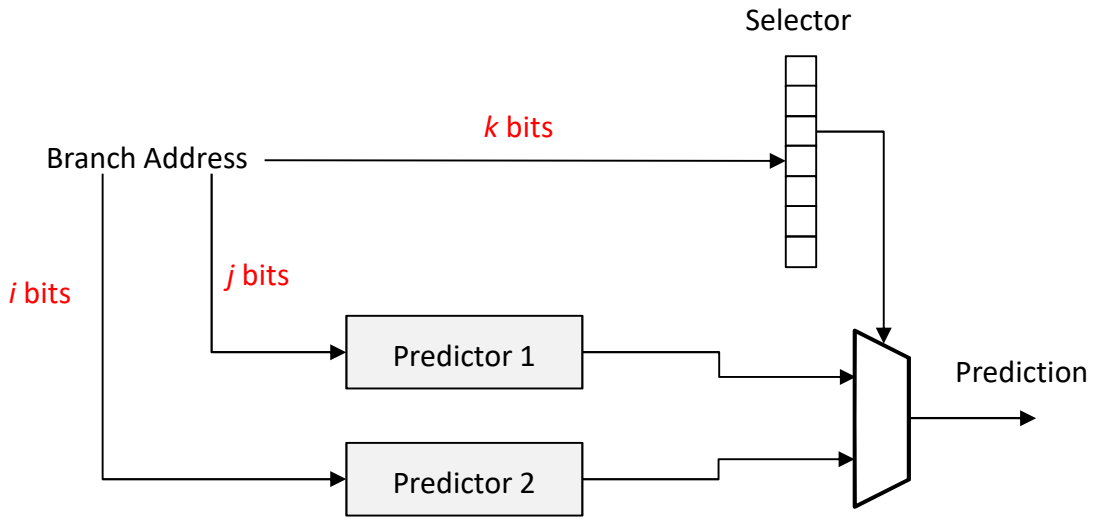
Two-level Branch Predictor (2/3)

- gshare: XOR address & history into the index bits
  - k-bit for global history would be a waste of registers
  - Global history and branch address can share the index bits using XOR operations



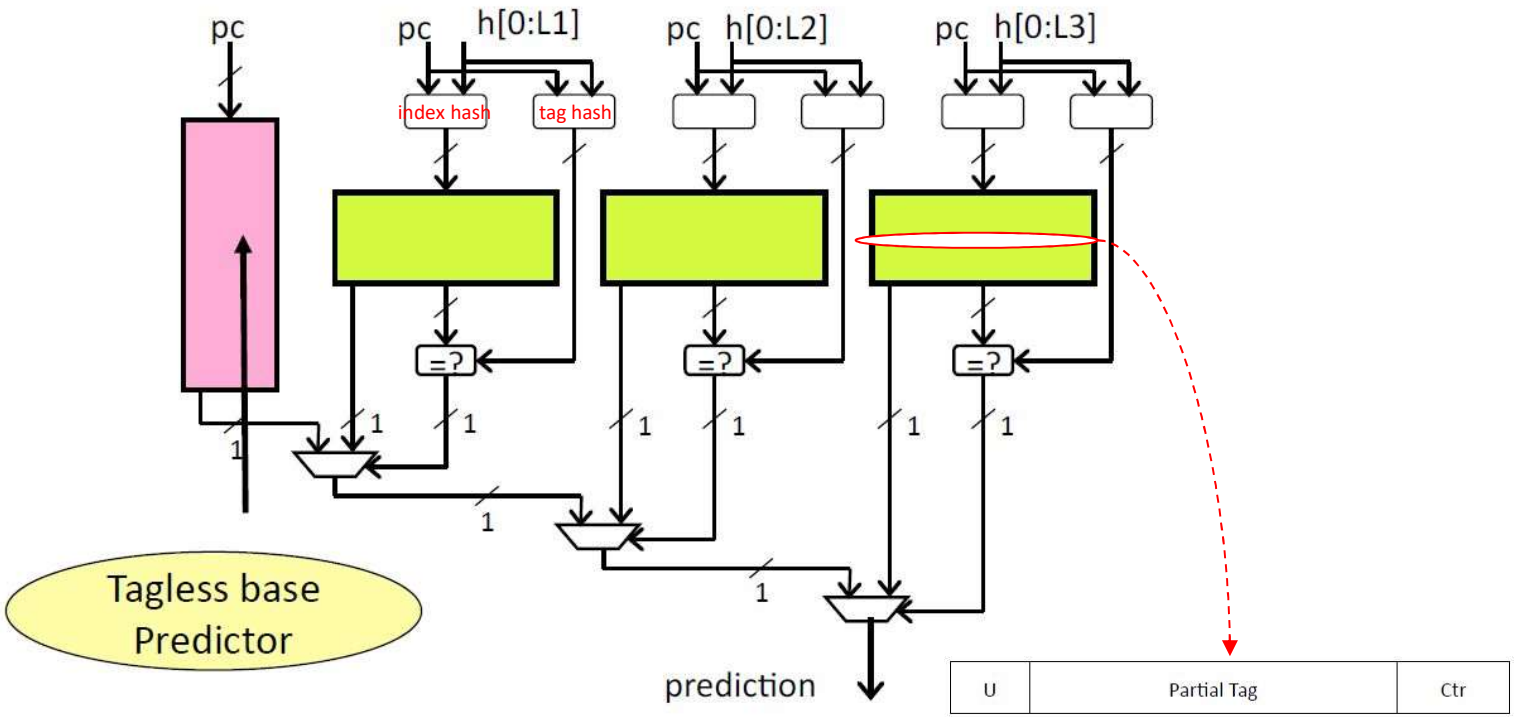
Two-level Branch Predictor (3/3)

- Another idea is to mix different predictors
  - The best BPU algorithm so far is the TAGE predictor
- Different predictors work for different code patterns
  - Multiple predictors can be used to adapt to different code sections in the program:



TAGE Branch Predictors (Multi-Level)

- The state-of-the-art branch predictor is TAGE:



André Seznec, "A New Case for the TAGE Branch Predictor," *MICRO 2011 : The 44th Annual IEEE/ACM Int. Symp. on Microarchitecture*, Dec. 2011.

Your Homework (1/2)

- Part 1: Analysis report (60% of the credit):
  - Study the branch predictor in Aquila
    - Disable BPU or change BHT size see what happens to CoreMark
  - Analyze the branch statistics (e.g. hit rate and miss rate for different types of branches) of CoreMark
  - Based on your analysis, discuss your plan on how to use a two-level predictor to improve the performance
  - You report on this part should be no more than two pages.

Your Homework (2/2)

- Part 2: Two-level predictor (40% of the credit)
  - Try out some ideas on two-level predictors
  - Extend your analysis report to include new results and new discussions. The overall report size should be no more than three pages.