HW#2 Branch Predictor Design

Chun-Jen Tsai
National Chiao Tung University
10/19/2021

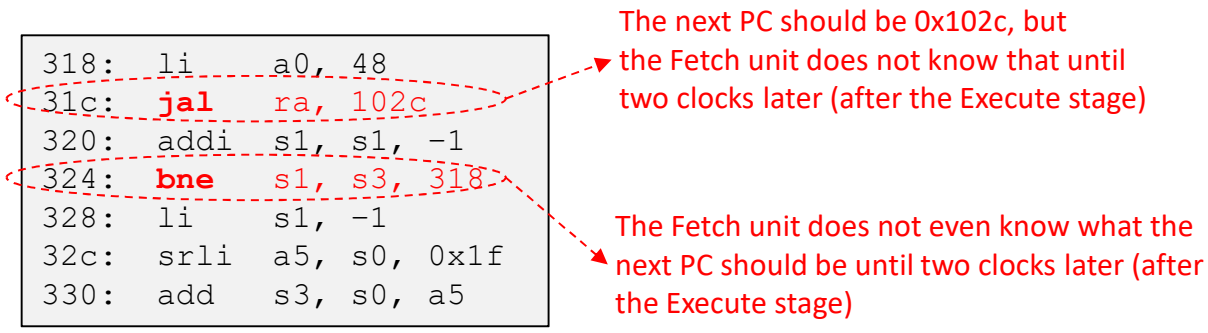
Homework Goal

- This homework requires you to modify the Branch Prediction Unit (BPU) of Aquila to improve performance
  - Change the parameters of the dynamic BPU in Aquila
  - Compare a static predictor to the dynamic branch predictor
  - Try other predictor ideas
- You have 2+1 weeks to implement this homework
  - You must upload your initial analysis report by 10/29, 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 11/8, 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 statiscs 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	?

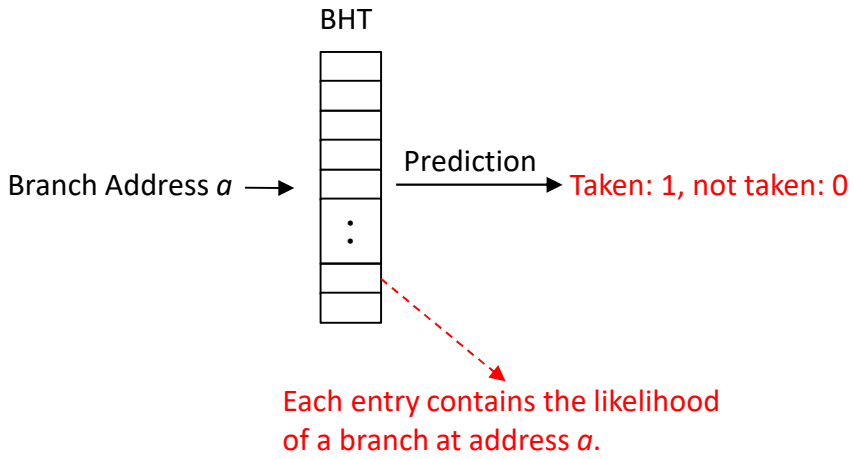
Next PC determined!

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

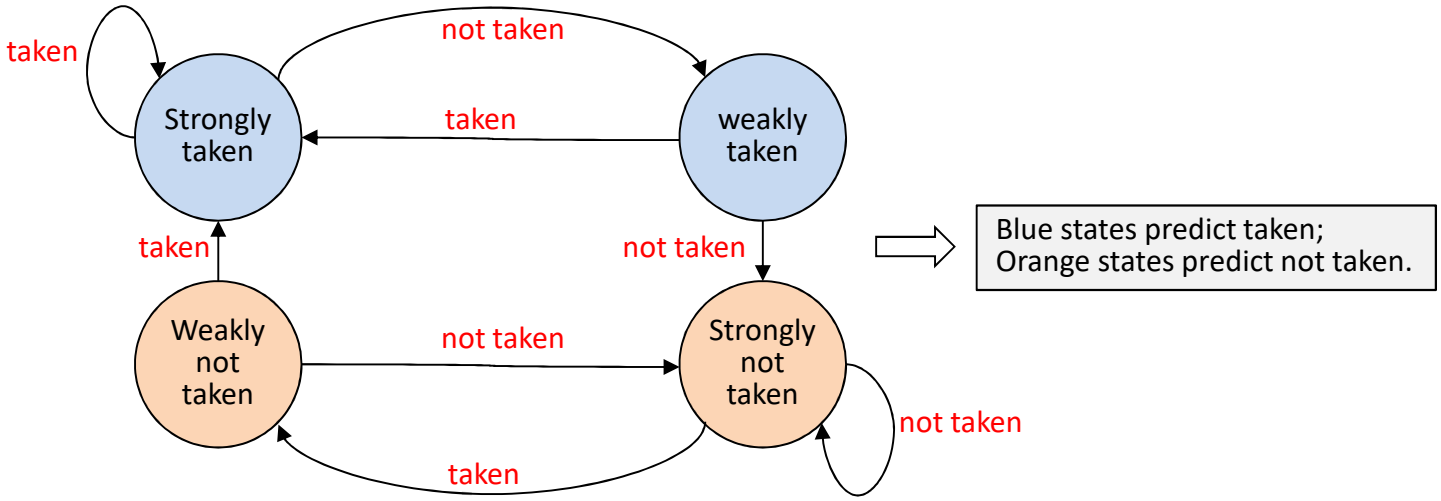
One-level Branch Predictor (1/2)

- 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:



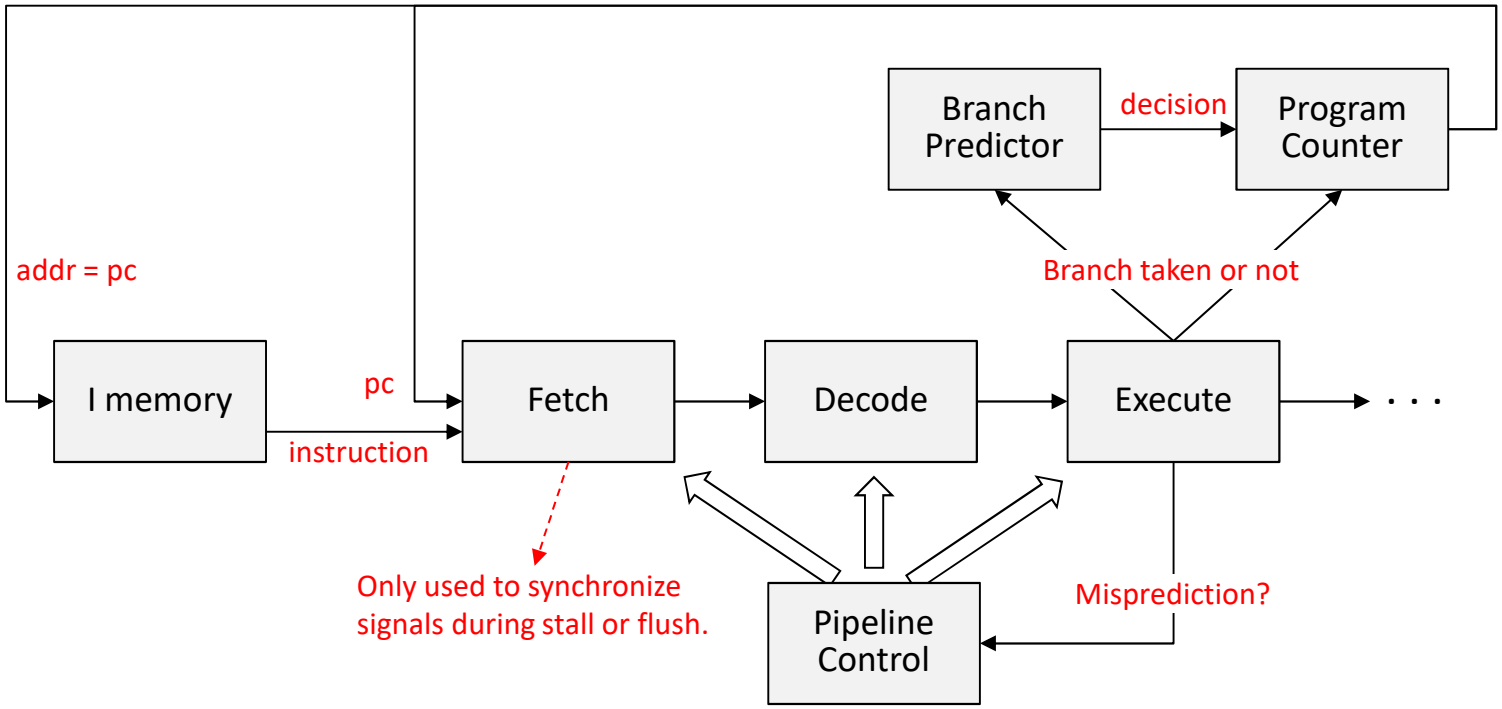
  - The state changes after the execute stage determines whether the branch is taken or not.

BHT Implementation

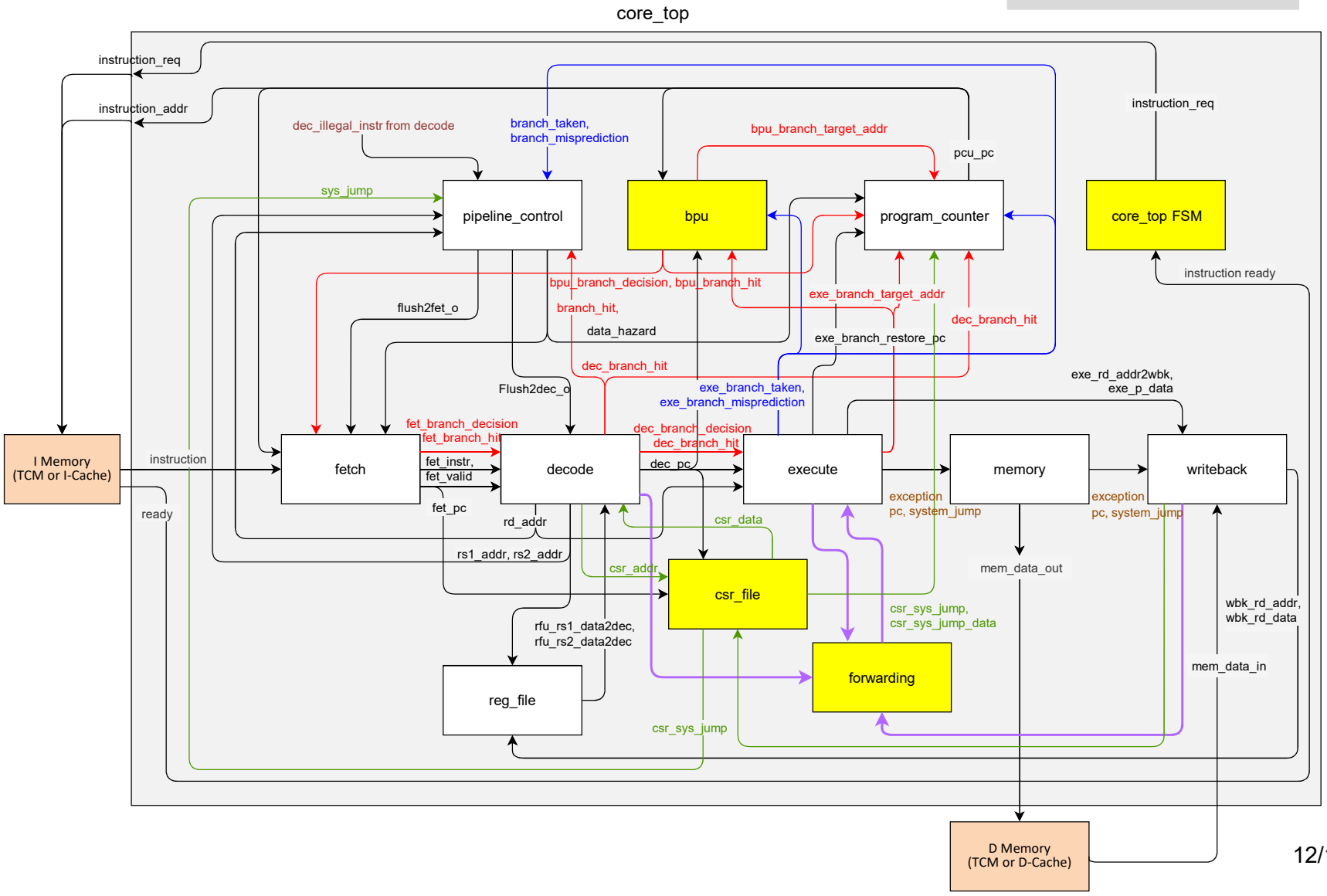
- The BHT is an associative memory
  - Data are retrieved using a TAG, instead of address
  - For BHT, TAG is the PC of the current branch instruction
- Theoretically, the BHT size should be large enough to accommodate all branch instructions in the program
- In Aquila, BHT size is specified by the parameter `ENTRY_NUM` in `bpu.v`
  - However, to actually change its size, you have to modify part of the Verilog code as well as the parameter.

Branch Prediction Flow in Aquila

- A branch predictor tells the fetch unit which instruction to fetch before the branch has been executed

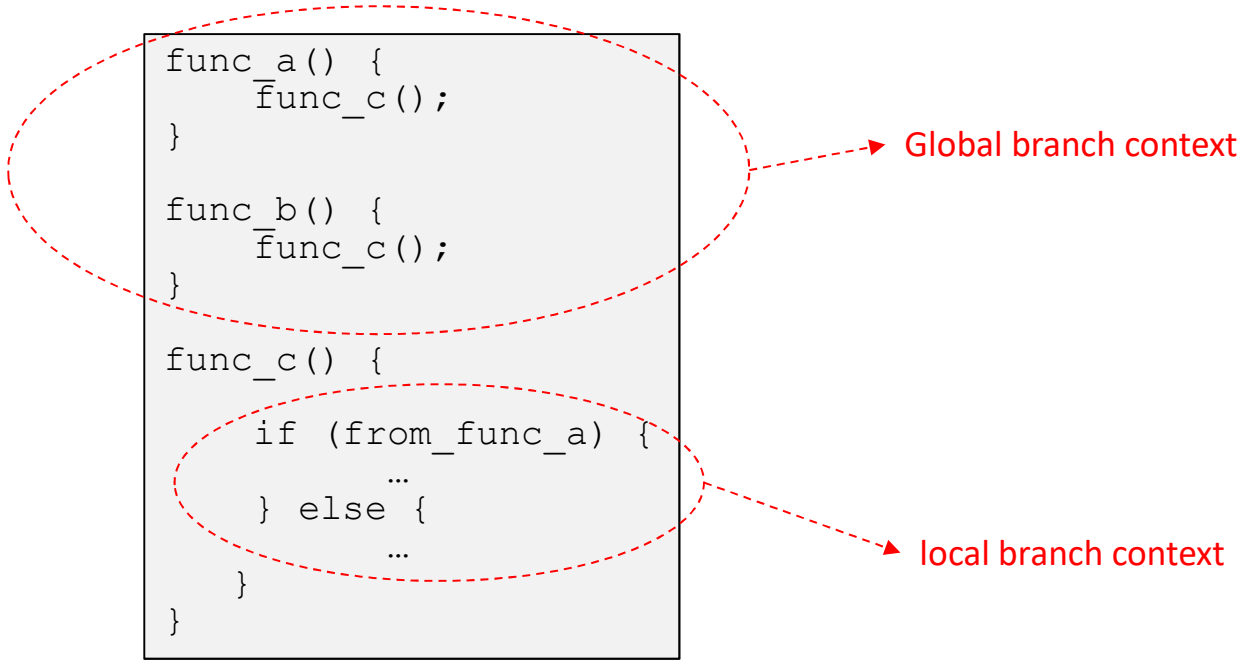


Aquila BPU Related Signals



Basic Ideas on Two-Level BPU

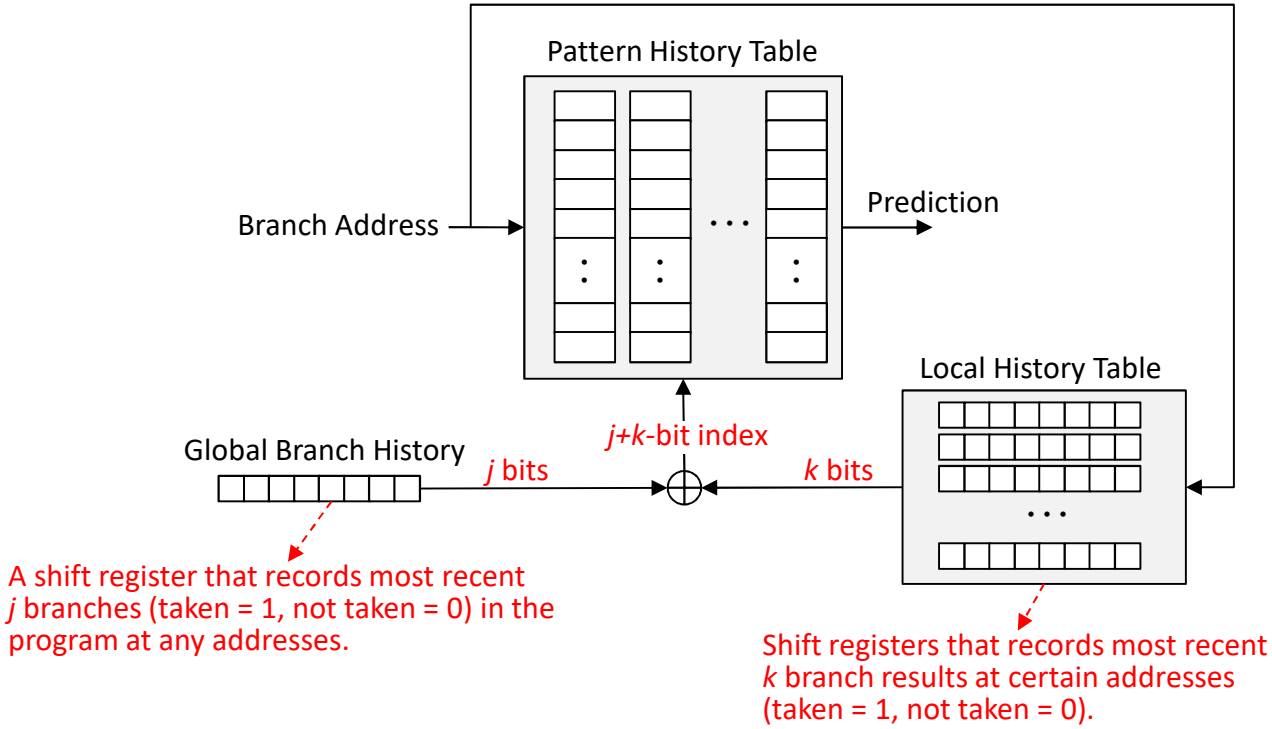
- One-level predictor lacks global context information



- Two-level branch predictor can take both global and local contexts into account

Two-level Branch Predictor (1/2)

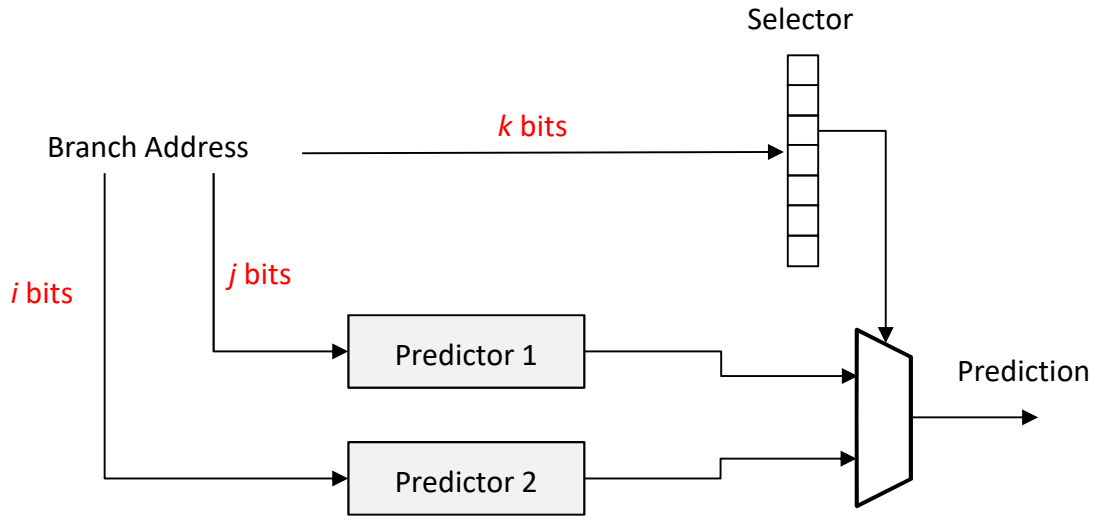
- A branch depends on the branch address as well as:
  - Nearby branches – recorded using Global Branch History
  - Longer history of the same branch – Local Branch History



T.-Y. Yeh and Y.N. Patt, "Two-level Training Branch Prediction," 24th ACM/IEEE Int. Symp. on Microarchitecture, Nov. 1991.

Two-level Branch Predictor (2/2)

- Different predictors work for different code patterns
  - Multiple predictors can be used to adapt to different code sections in the program:



  - You may have to set `Number_Of_Runs` in `dhry_1.c` to a large number to see the improvement of a complex BPU

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

Your Homework (1/2)

- Part 1: Analysis report (60% of the credit):
  - Study the branch predictor in Aquila
  - Change the BHT size from 32 to 64 and see if the DMIPS is increased
  - Analyze the branch statistics (e.g. hit rate and miss rate for different types of branches) of Dhrystone
  - You can also design your own benchmark program to show the weakness of the BPU in Aquila
  - Based on your analysis on existing Aquila BPU, 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
  - Note that you can design your own benchmark program to show the advantage of your two-level predictor. For this part, you don’t have to use Dhrystone
  - Extend your analysis report to include new results and new discussions. The overall report size should be no more than three pages.