Lecture-10 (Branch Prediction)
CS422-Spring 2018

Biswa@CSE-IITK

PA2: ICBP

- In-class Branch Prediction Championship
- Will be evaluated based on MPPKI – Miss Prediction Per Kilo instructions (10 traces)
- Goal: Design a predictor that is better than Ghsare based predictors.

- Bonus: 10/5/2 (Top 1/2/3 submissions)

Few Subtle Issues

- Is the order of update to PHT and BHT important ?
- What about nested branches: New branch while handling the current branch: Is it possible in 5 stage pipeline? What about 10-stage, 20-stage ?
- Aliasing
- PC = PC + 4 ?
  PC = EX Stage Target
  PC based on BTB?

Global Branch Correlation

- Recently executed branch outcomes in the execution path is correlated with the outcome of the next branch

- If first branch not taken, second also not taken

- If first branch taken, second definitely not taken

Global Branch Correlation

```
branch Y: if (cond1)
...
branch Z: if (cond2)
...
branch X: if (cond1 AND cond2)
```

- If Y and Z both taken, then X also taken
- If Y or Z not taken, then X also not taken

Global Branch Correlation

- Eqntott, SPEC 1992

```
if (aa!=2)                    ;; B1
      aa=0;
if (bb!=2)                    ;; B2
      bb=0;
if (aa==bb) {           ;; B3
      ....
 }
```

If B1 is not taken (i.e., aa==0@B3) and B2 is not taken (i.e. bb=0@B3) then B3 is certainly taken

Interference

- Sharing the PHTs between histories/branches leads to interference
  - Different branches map to the same PHT entry and modify it
  - Interference can be positive, negative, or neutral
- Interference can be eliminated by dedicating a PHT per branch
  -- Too much hardware cost
- How else can you eliminate or reduce interference?

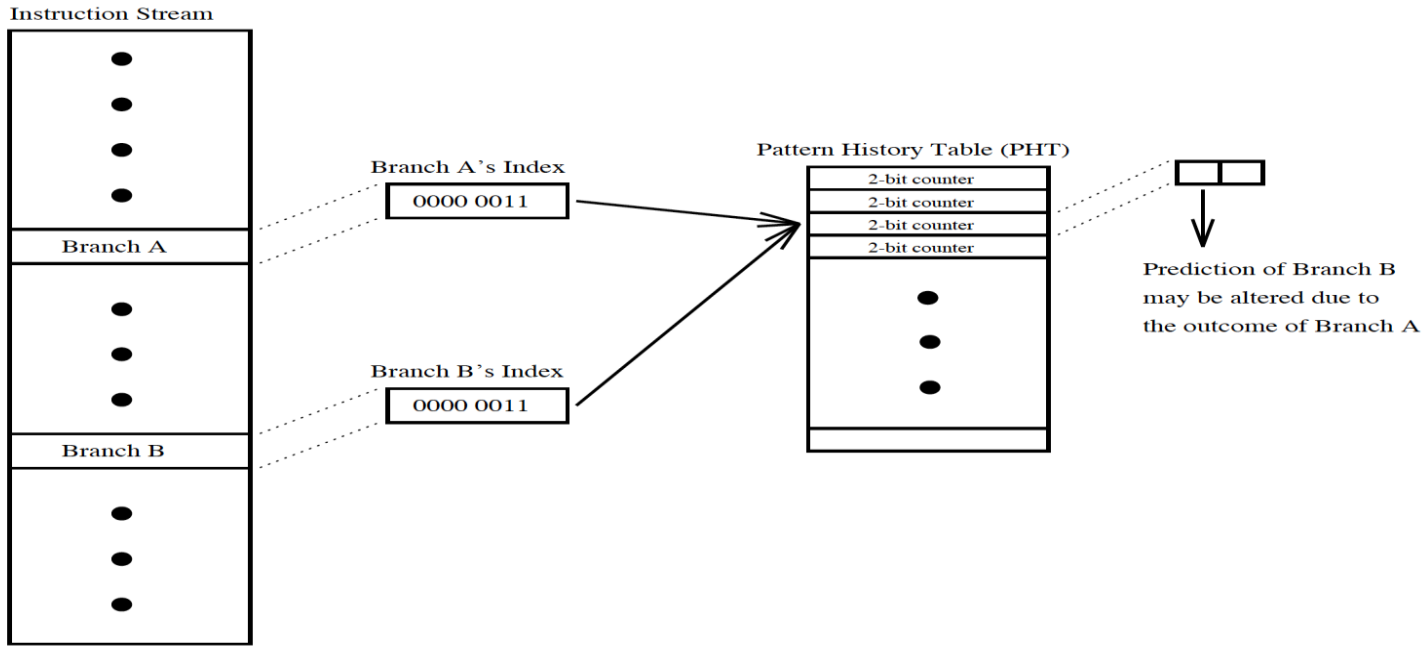


Figure 2: Interference in a two-level predictor.

Reducing Interference

- Increase size of PHT
- Branch filtering
  - Predict highly-biased branches separately so that they do not consume PHT entries
  - E.g., static prediction or BTB based prediction
- Hashing/index-randomization
  - Gshare
  - Gskew
- Agree prediction

Biased Branches

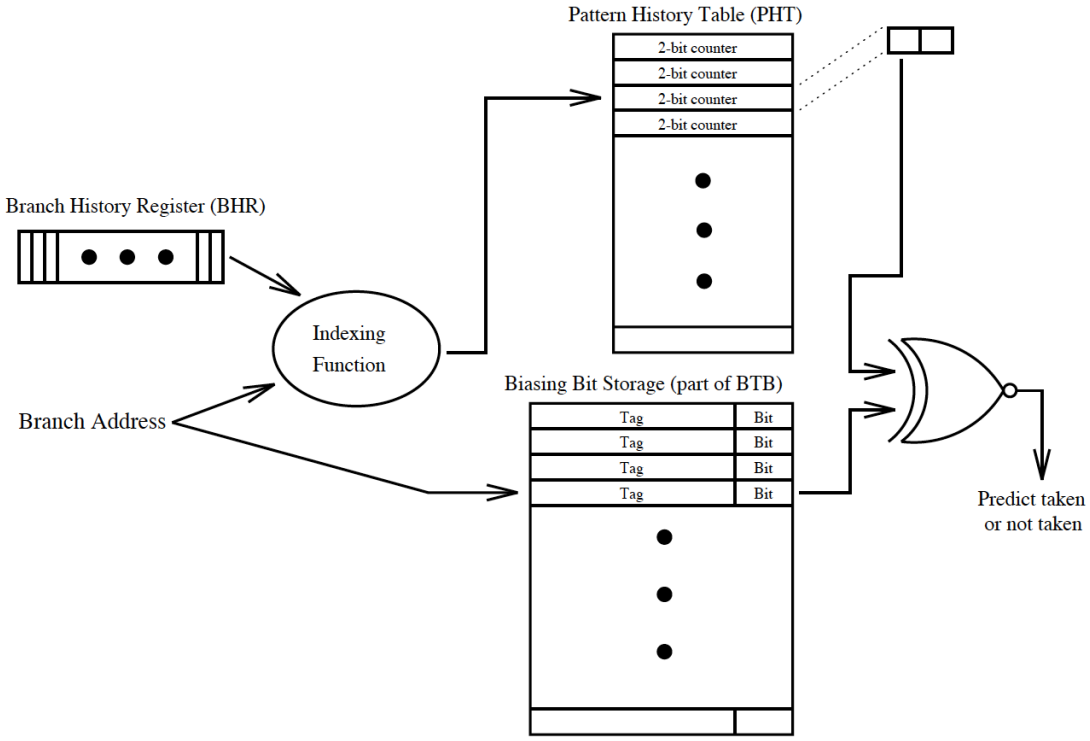
- Observation: Many branches are biased in one direction (e.g., 99% taken)
- Problem: These branches *pollute* the branch prediction structures → make the prediction of other branches difficult by causing "interference" in branch prediction tables and history registers
- Solution: Detect such biased branches, and predict them with a simpler predictor (e.g., last time, static, …)
- Chang et al., "Branch classification: a new mechanism for improving branch predictor performance," MICRO 1994.

Agree Predictor [ISCA '97]

- Idea 2: Agree prediction
  - Each branch has a "bias" bit associated with it in BTB
    - Ideally, most likely outcome for the branch
  - High bit of the PHT counter indicates whether or not the prediction agrees with the bias bit (not whether or not prediction is taken)

+ Reduces negative interference (Why???)

-- Requires determining bias bits (compiler vs. hardware)

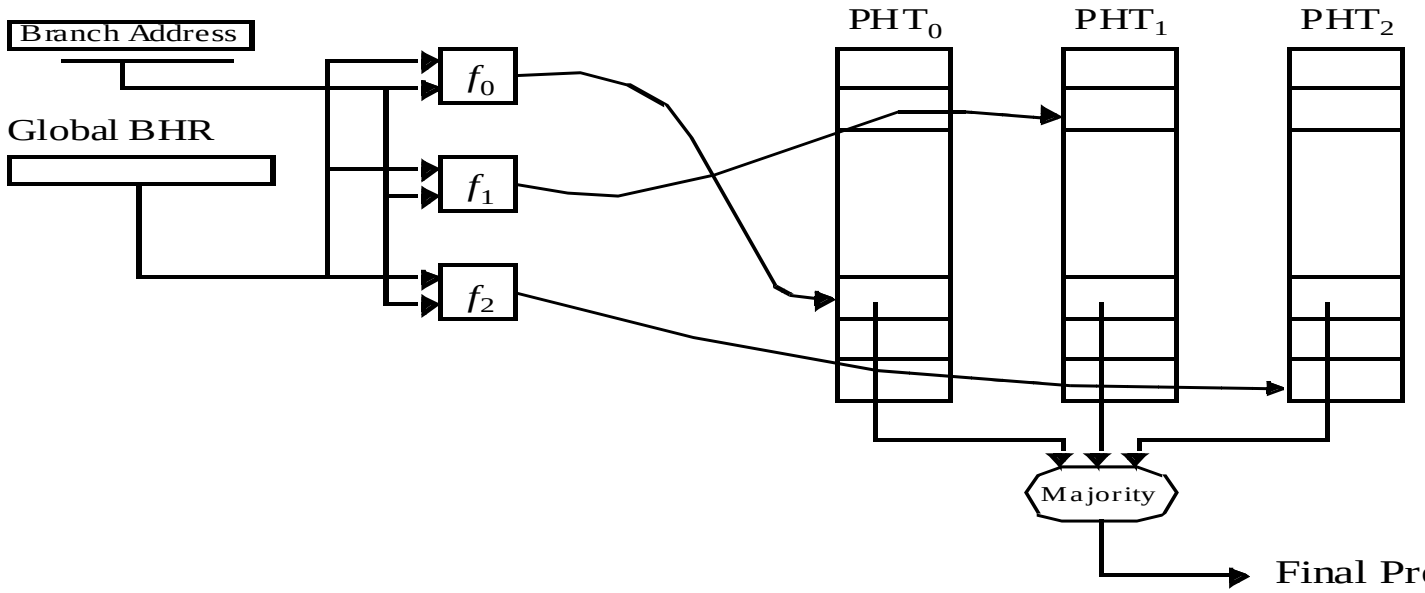


GSkew

- Idea 3: Gskew predictor
  - Multiple PHTs
  - Each indexed with a different type of hash function, Final prediction is a majority vote

+ Distributes interference patterns in a more randomized way (interfering patterns less likely in different PHTs at the same time)

-- More complexity (due to multiple PHTs, hash functions)



Seznec, "An optimized 2bcgskew branch predictor," IRISA Tech Report 1993.

Michaud, "Trading conflict and capacity aliasing in conditional branch predictors," ISCA 1997

Agree Predictor

- Assume two branches have taken rates of 85% and 15%.
- Assume they conflict in the PHT

- Let's compute the probability they have opposite outcomes
  - Baseline predictor:
    - P (b1 T, b2 NT) + P (b1 NT, b2 T)

      = (85%*85%) + (15%*15%) = 74.5%
  - Agree predictor:
    - Assume bias bits are set to T (b1) and NT (b2)
    - P (b1 agree, b2 disagree) + P (b1 disagree, b2 agree)

      = (85%*15%) + (15%*85%) = 25.5%

- Works because most branches are biased (not 50% taken)

Additional Reading: Intel Technology Journal ['03]

The advanced branch prediction in the Pentium M processor is based on the Intel Pentium® 4 processor's [6] branch predictor. On top of that, two additional predictors to capture special program flows, were added: a Loop Detector and an Indirect Branch Predictor.

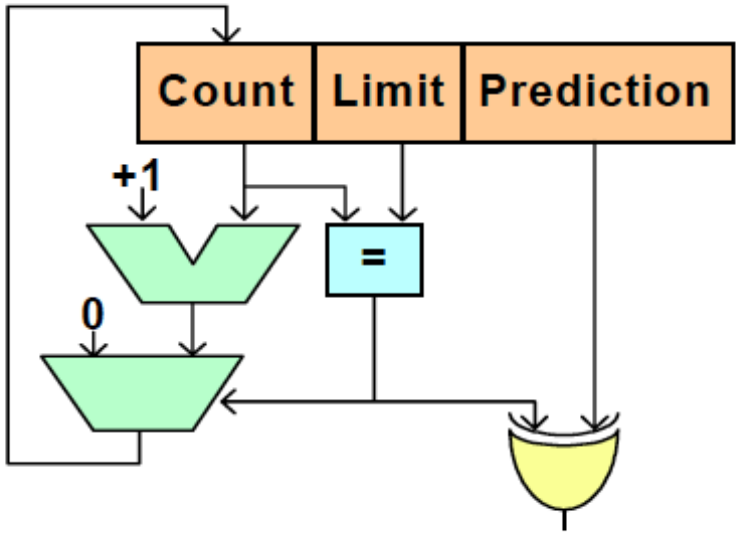


Figure 2: The Loop Detector logic

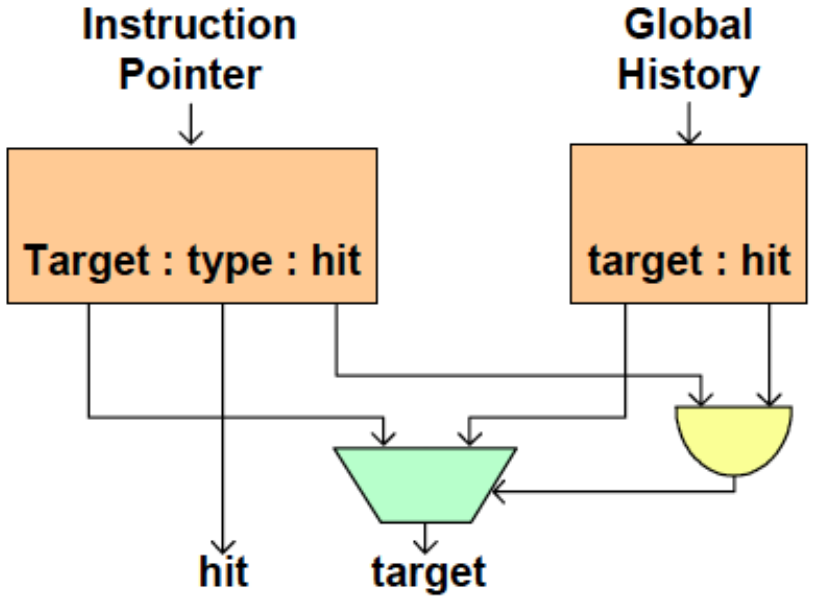
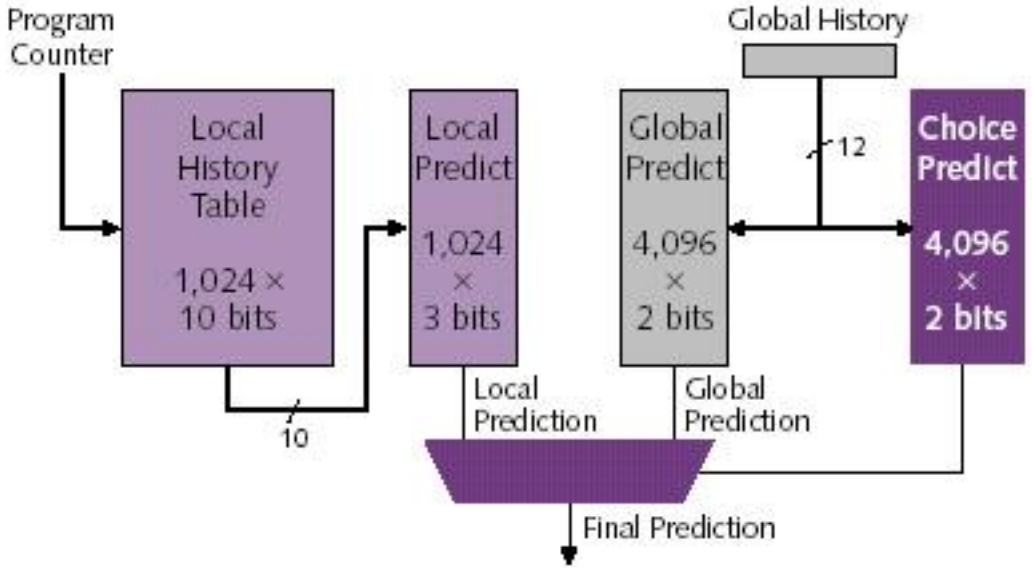
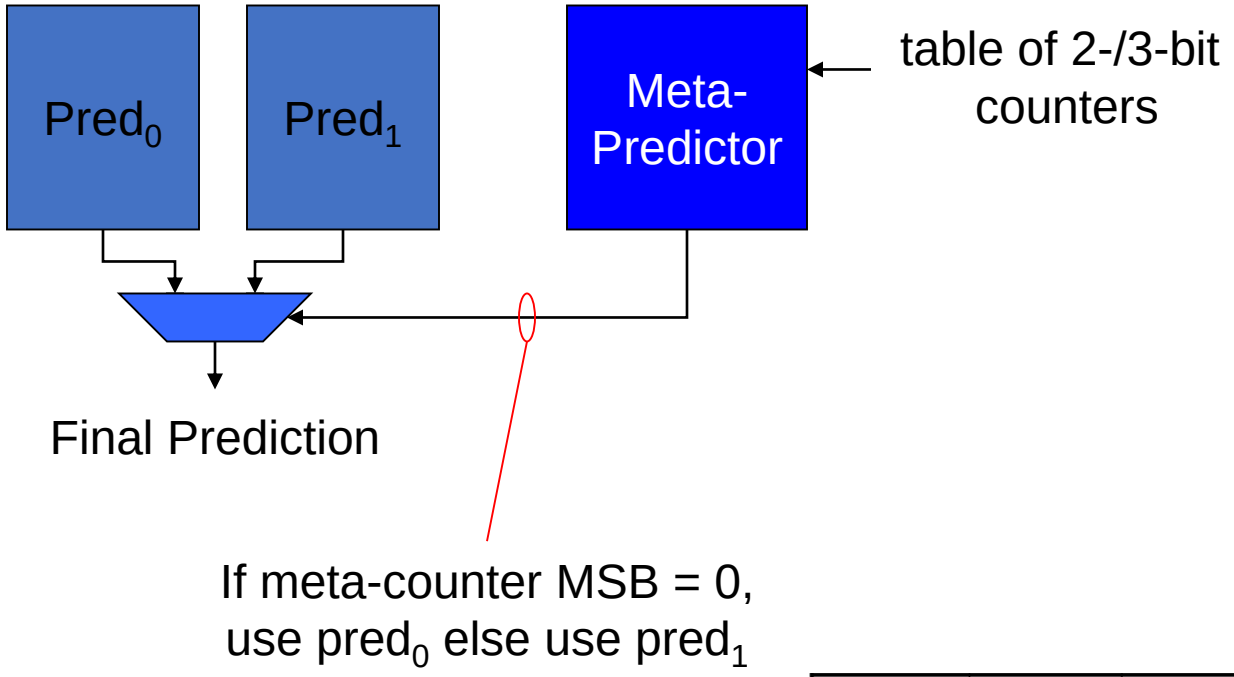


Figure 3: The Indirect Branch Predictor logic

Online Test 3 but not in-class

Moving Forward: ALPHA 21264 (Tournament Predictor) [IEEE MICRO '99]

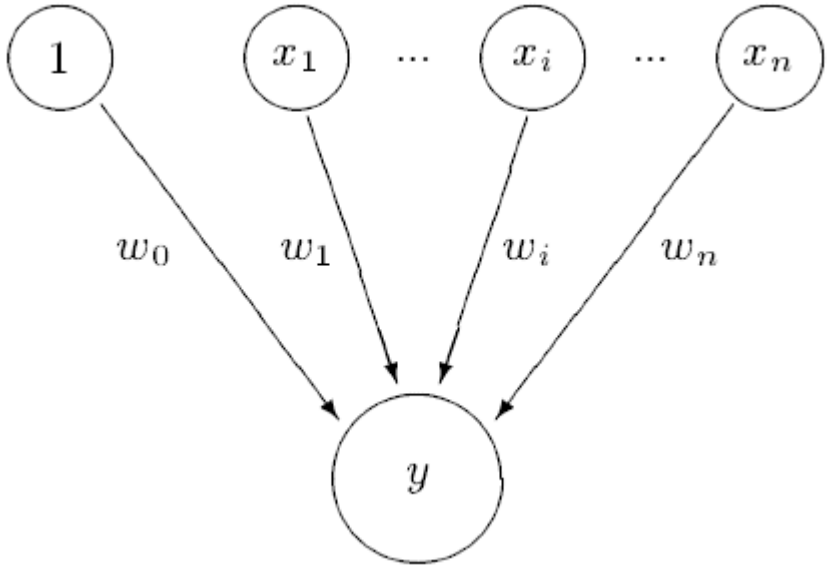




$Pred_0$	$Pred_1$	Meta Update
✗	✗	---
✗	✓	Inc
✓	✗	Dec
✓	✓	---

Perceptron Branch Predictor [HPCA '01]

- Idea: Use a perceptron to learn the correlations between branch history register bits and branch outcome
- A perceptron learns a target Boolean function of N inputs

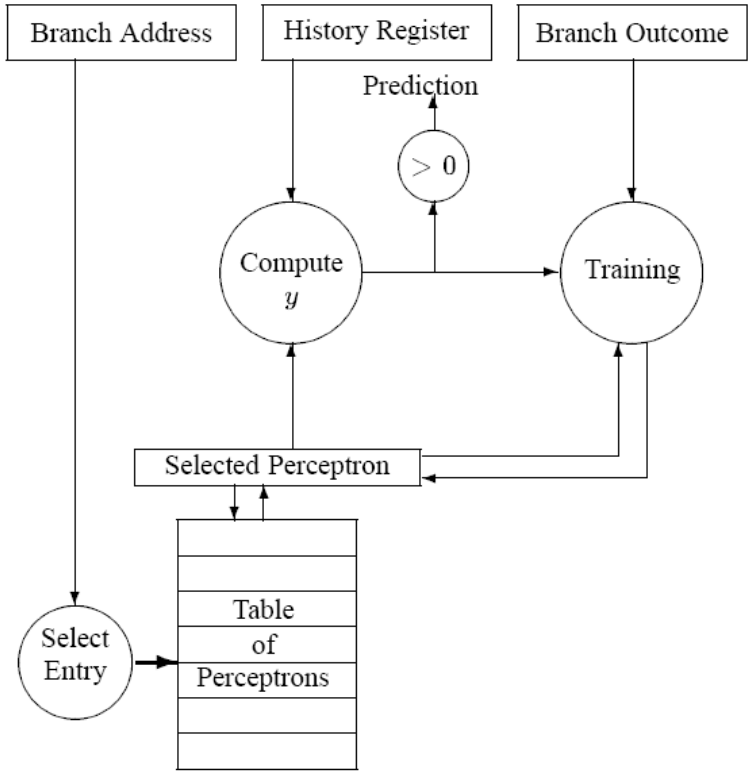


$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$

A perceptron contains a set of weights wi
→ Each weight corresponds to a bit in the GHR
→How much the bit is correlated with the direction of the branch
→ Positive correlation: large + weight
→ Negative correlation: large - weight

Prediction:
→ Express GHR bits as 1 (T) and -1 (NT)
→ Take dot product of GHR and weights
→ If output > 0, predict taken

Perceptron Branch Predictor



Prediction function:

Dot product of GHR and perceptron weights

$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$

Output compared to 0

Bias weight (bias of branch independent of the history)

Training function:

```
if sign(y_out) ≠ t or |y_out| ≤ θ then
        for i := 0 to n do
                w_i := w_i + t x_i
        end for
end if
```