

---

# Topic 14: Dealing with Branches

COS / ELE 375

Computer Architecture and Organization

Princeton University  
Fall 2015

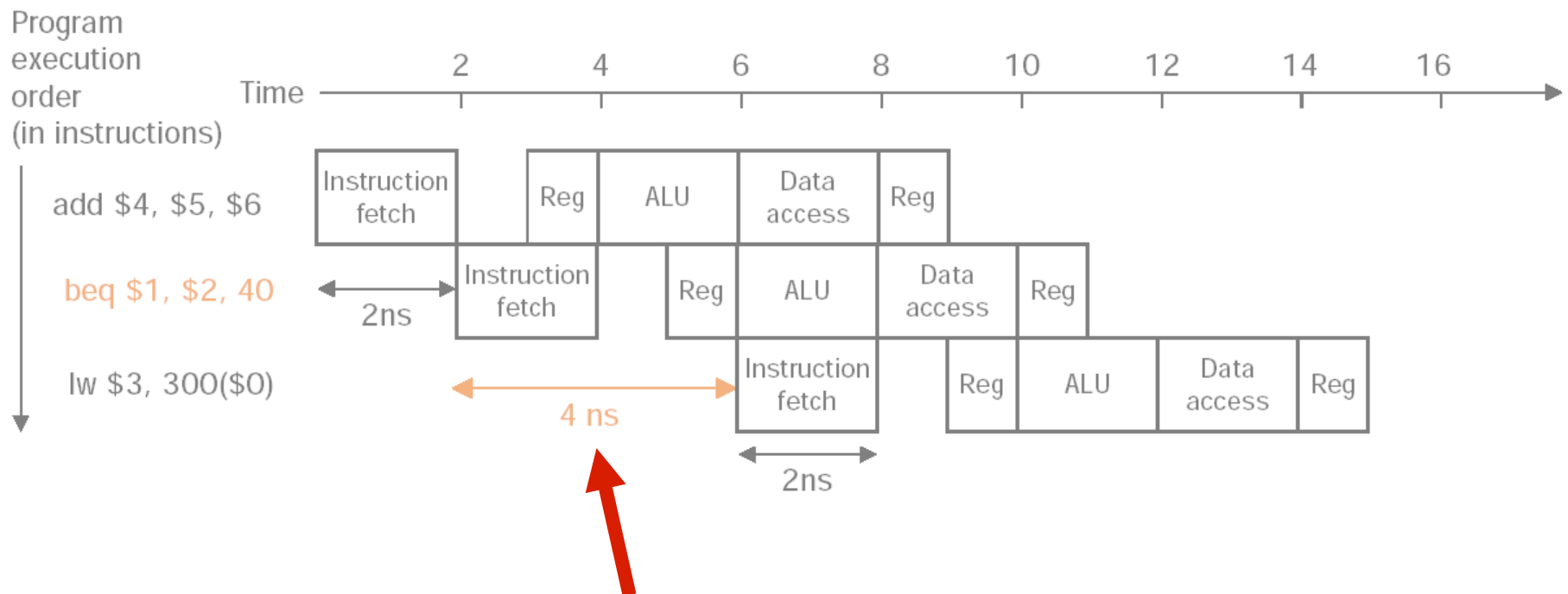
Prof. David August

# FLASHBACK: Pipeline Hazards

## Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Stall, Predict, or Delay:



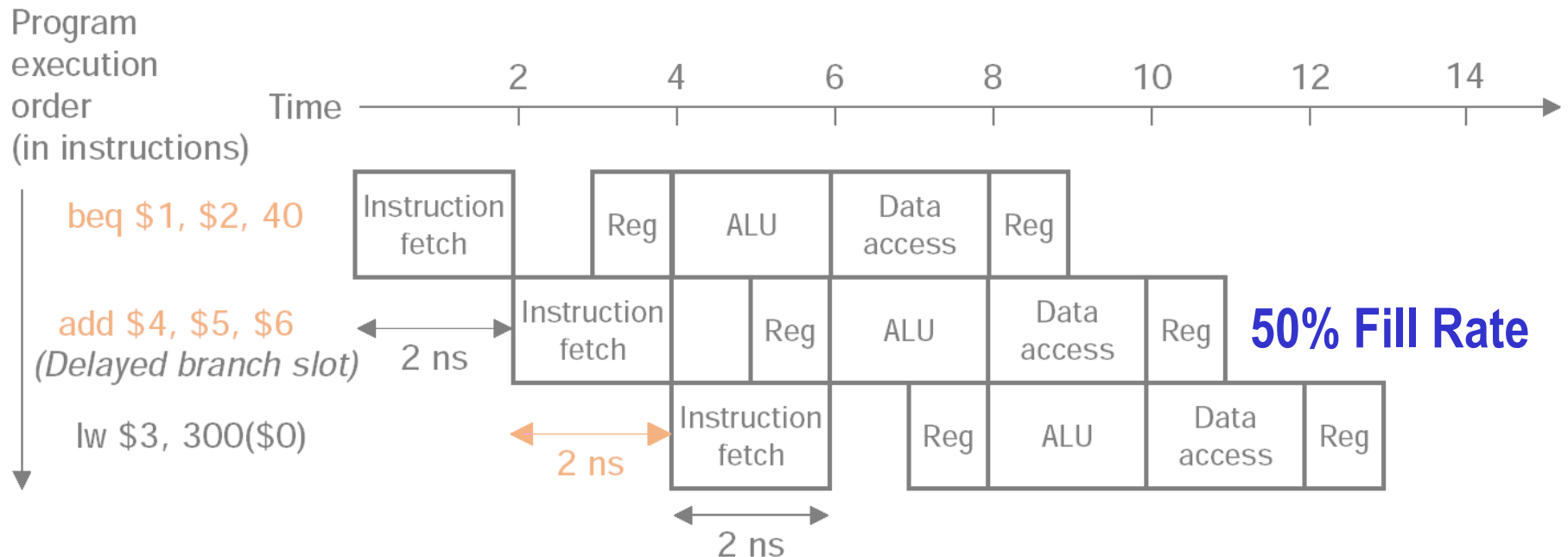
Pipeline Stall - only 1 cycle/stage delay...

# FLASHBACK: Pipeline Hazards

## Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

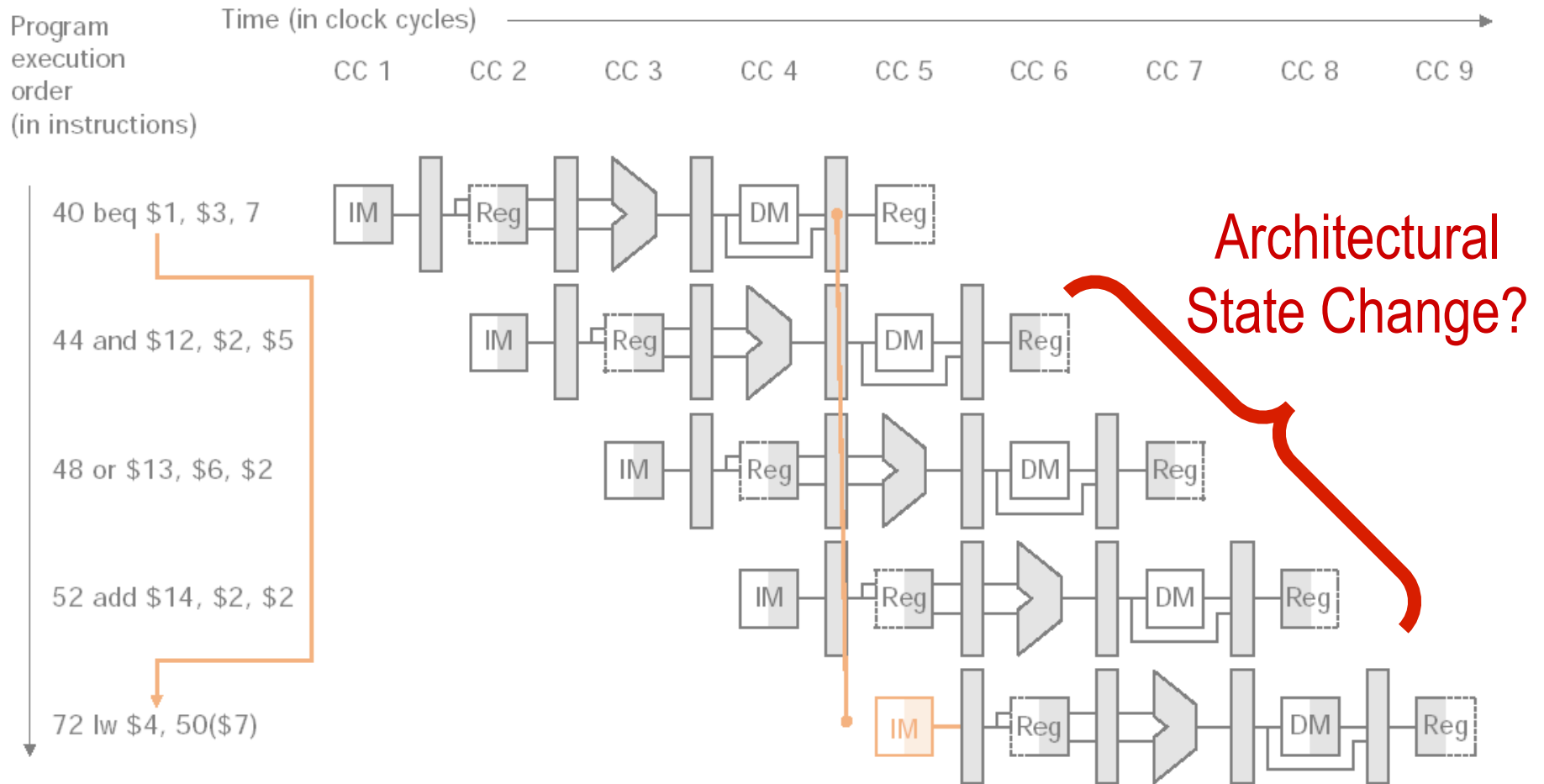
### Delayed Decision (Used in MIPS):



More about Branch Prediction/Delayed Branching Later...

# FLASHBACK: What About Control Hazards?

## (Predict Not-Taken Machine)



**We are OK, as long as we squash. Can we reduce delay?**

# Cold Realities

---

Pipelines are much longer (Pentium 4 has +22 stages!)  
1 branch delay cycle not possible in these designs

## Stall

- Makes a Pentium 4 run about as fast as an 80486
- What is the point of pipelining if it is all fill/drain?

## Branch Delay Slots

- Very rare to be able to fill more than 1 slot
- Ugly architecture - exposes implementation

Delay slots for a processor that can handle multiple instructions simultaneously?

# Cold Realities

---

Pipelines are much longer (Pentium 4 has +22 stages!)

Stall - Nope

Branch Delay Slots - Nope

Predict Not Taken

- Roughly 33% of branches are taken

Is a 67% prediction rate good enough?

# The Importance of High Prediction Rates

---

- In general purpose codes, 1 in every 4 or 5 instructions is a branch.
- Misprediction penalties are high
  - 17 wasted cycles in Pentium 4
  - 7 wasted cycles in Alpha 21264

Example:

- Pipelining throughput (ideal) = 1 CPI
- 67% prediction rate, 1 in 4 instructions a branch
- 20 cycle penalty

$$0.25 \text{ br/i} * 0.33 \text{ miss/br} * 20 \text{ c/miss} + 1 \text{ c/i} = 2.67 \text{ c/i}$$

$$\text{Slowdown} = 2.67x$$

# The Importance of High Prediction Rates

---

- In real non-scientific codes, 1 in every 4 or 5 instructions is a branch.
- Misprediction penalties are high
- Modern machines execute multiple instructions per cycle

Wide-Issue Example:

- Pipelining throughput (ideal) = 4 IPC (0.25 CPI)
- 67% prediction rate, 1 in 4 instructions a branch
- 20 cycle penalty

$$0.25 \text{ br/i} * 0.33 \text{ miss/br} * 20 \text{ c/miss} + 0.25 \text{ c/i} = 1.9 \text{ c/i}$$

$$\text{Slowdown} = 7.6x$$

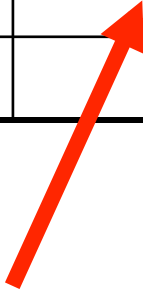


# How do we get better prediction rates?

## Better Prediction

- Predictions can be made by the **Hardware** or by the **Compiler**
- Predictions can set before execution (**Static**) or adapt during execution (**Dynamic**)

	Hardware	Compiler
Static	Predict Not Taken	
Dynamic		



Let's Look Here Next For No Apparent Reason

# Compiler-Static Prediction

---

Key idea: Compiler indicates a prediction to the HW

## Communication Schemes

- Use a **prediction bit** in your branch encoding
  - Bit = 1 → predict taken
  - Bit = 0 → predict not-taken
- Backward taken, forward not taken (BTFNT)
  - Negative displacement → predict taken
  - Positive displacement → predict not-taken

Why BTFNT and not FTBNT?  
Does it really matter?

# Compiler-Static Prediction

---

Key idea: Compiler indicates a prediction to the HW

## Compiler Prediction Methods

- Profiling
  - Run program and observe

Issues?

- Rule-based
  - Loops do
  - Exits don't

Why?

# How do we get better prediction rates?

## Better Prediction

- Predictions can be made by the **Hardware** or by the **Compiler**
- Predictions can set before execution (**Static**) or adapt during execution (**Dynamic**)

	Hardware	Compiler
Static	Predict Not Taken	Prediction Bit, BTFNT
Dynamic		

Advantages of these quadrants?  
Disadvantages?

# Dynamic Branch Prediction

---

Key idea: Branches have personality

Question: What do you think the branch will do next?  
(1 = taken, 0 = not taken)

Branch History: 111 ?

Branch History: 101010101 ?

Branch History: 110011001100110011 ?

Branch History: 00000000001111111111 ?

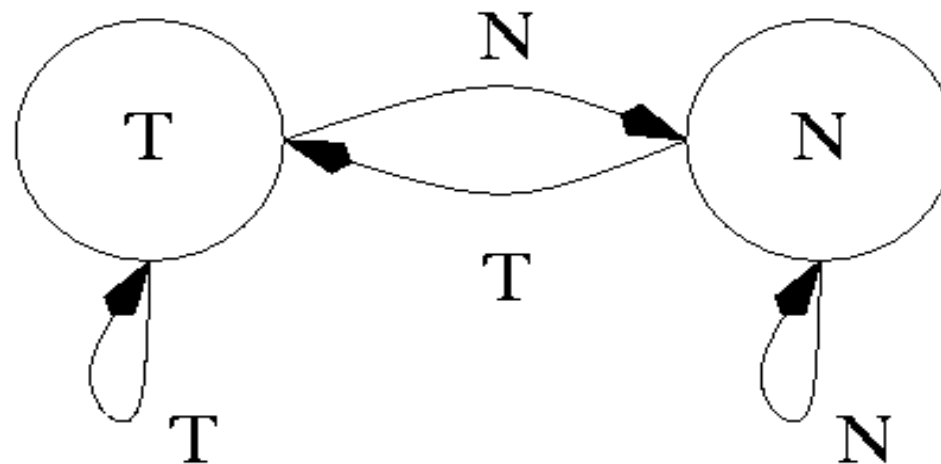
Branch History: 111111101111 ?

How Do We Capture This?

# Hardware-Dynamic Branch Prediction Automata

Consider history of 111111 or 000000

**Automaton: Last-Time (LT)**



**When is a Last-Time predictor better than a compiler static predictor?  
What happens the first time the branch is seen?**

# Hardware - Dynamic Branch Prediction

---

## Automata - More States → Better

### Other Automata

Consider another pattern with Last-Time:

- 1 1 1 0 1 1 1 0 1 1 1 0 1
- Last-Time = 6 misses

Only 3 misses with 4 states:

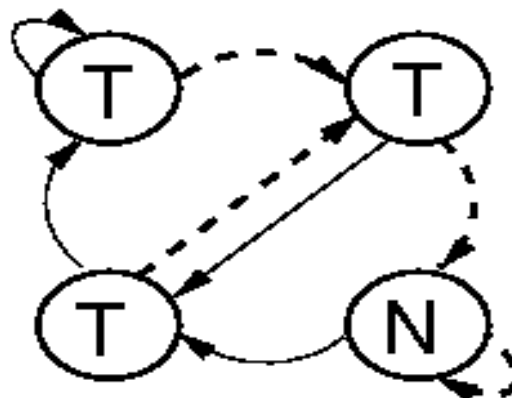
State	Meaning	Prediction
00	Strongly Not Taken	Not Taken
01	Weakly Not Taken	Not Taken
10	Weakly Taken	Taken
11	Strongly Taken	Taken

- Used by Alpha 21164, Sun UltraSPARC, MIPS R10000, and others.
- Jim Smith, 1991

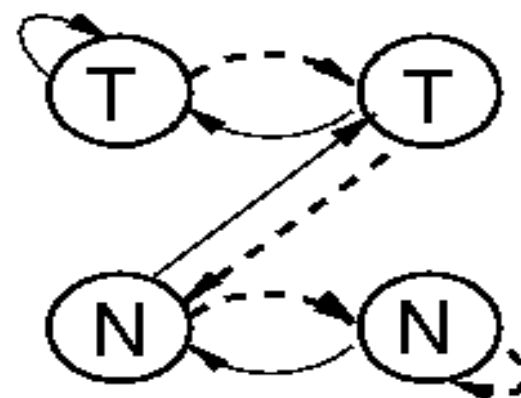
# Automata



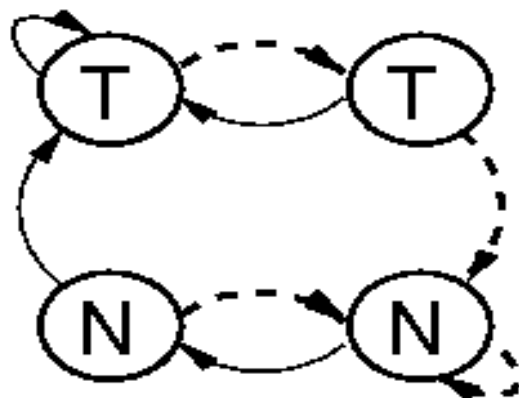
Automaton Last-Time (LT)



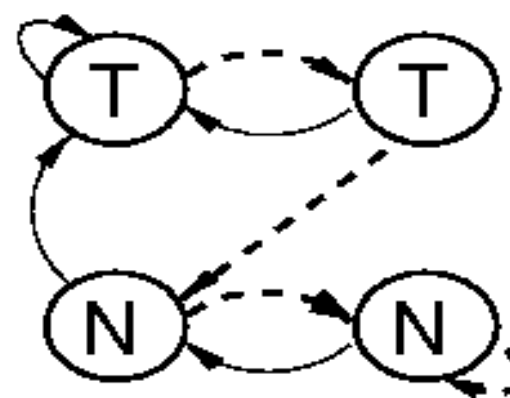
Automaton A1



Automaton A2  
(2-bit Saturating Up-down Counter)



Automaton A3



Automaton A4



# Branch Correlation

---

- All these automata capture only *Self-Correlation* of branches.
- Branches can be correlated: (From *eqntott*, SPEC92)

```
if (aa == 2)          /* branch 1 */
    aa = 0;
if (bb == 2)          /* branch 2 */
    bb = 0;
if (aa != bb) {       /* branch 3 */
    . . . . .
}
```

- If the conditions of branch 1 and branch 2 are TRUE, the condition of branch 3 is FALSE.

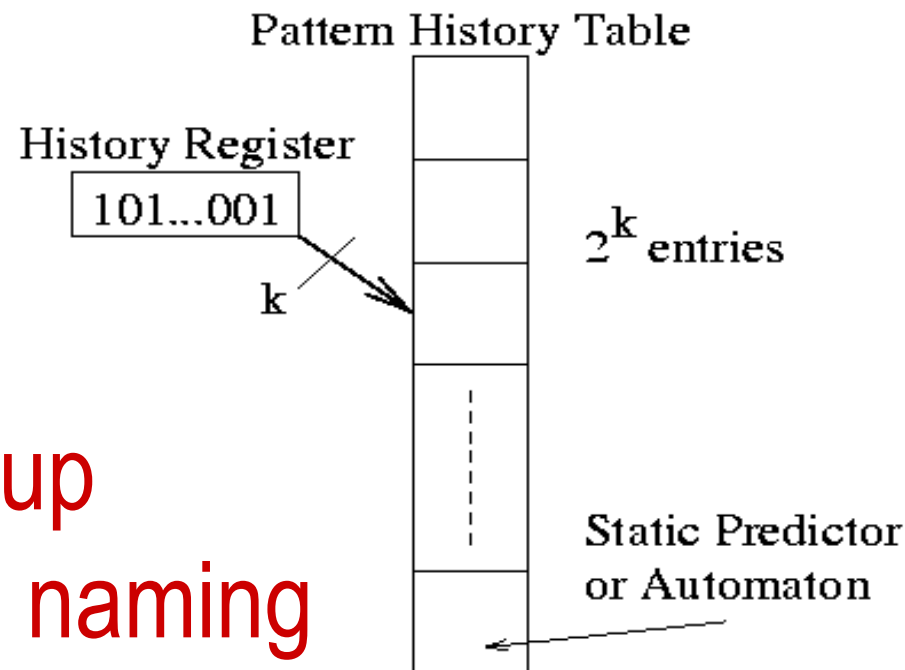
**Can other-branch history be used to improve prediction?**

# Branch Prediction with History Tables

## Types of Pattern History Tables

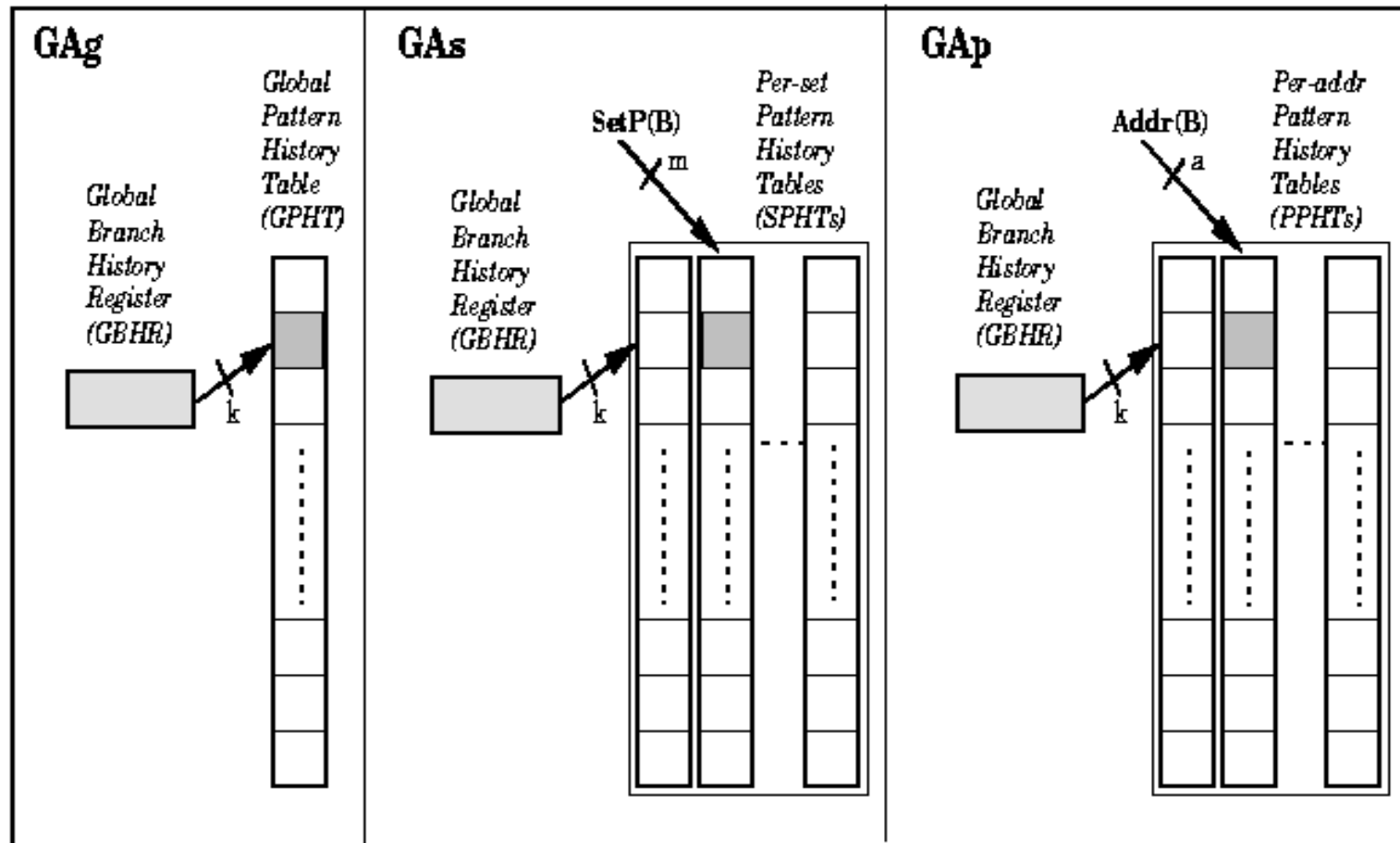
Pattern History Tables can be:

- *Global* - one predictor for each pattern shared with all branches.
- *Per-address* - one predictor for each pattern for each branch.
- *per-Set* - one predictor for each pattern for each set of branches.

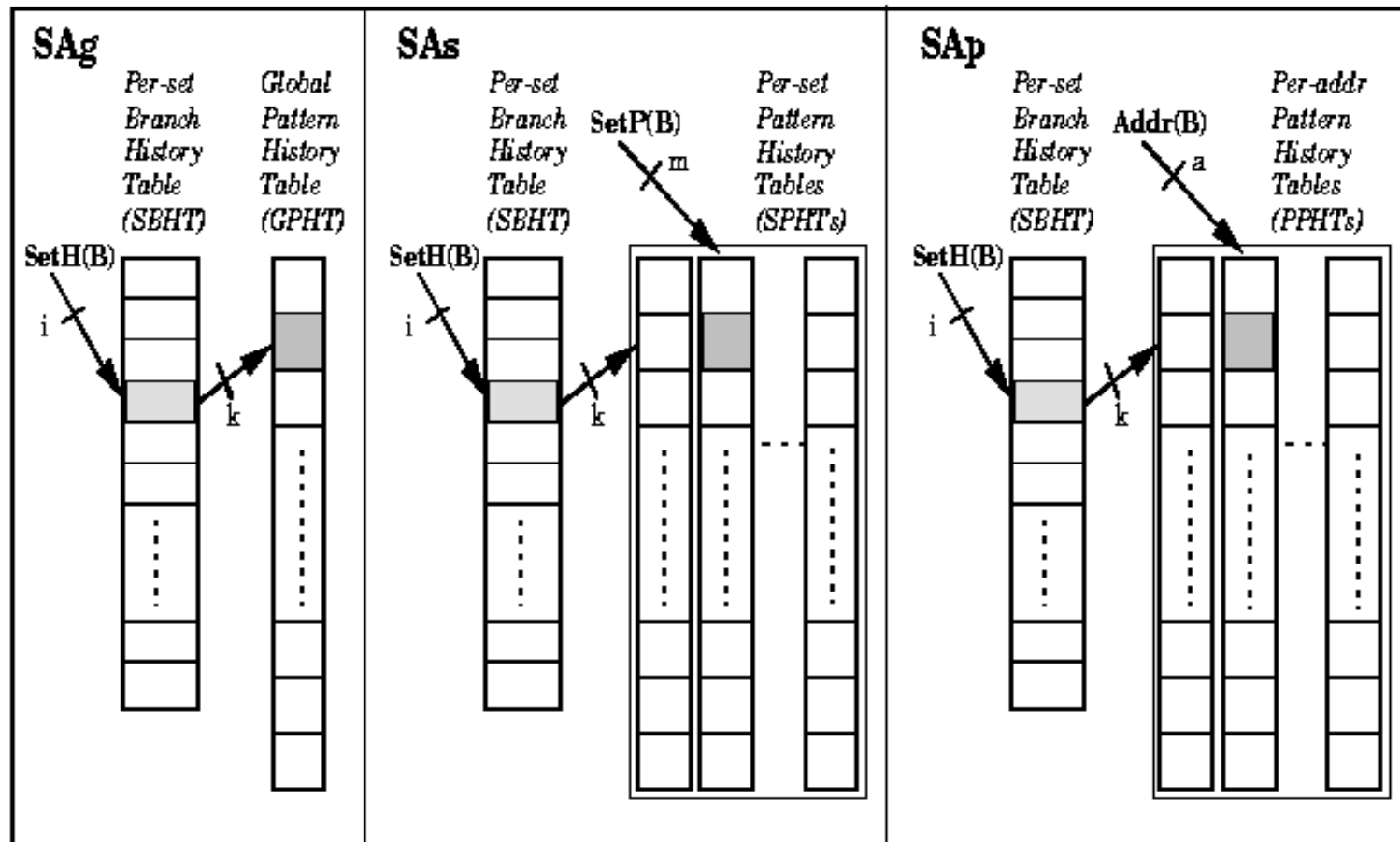


Warm-up  
[GPS]A[gps] naming

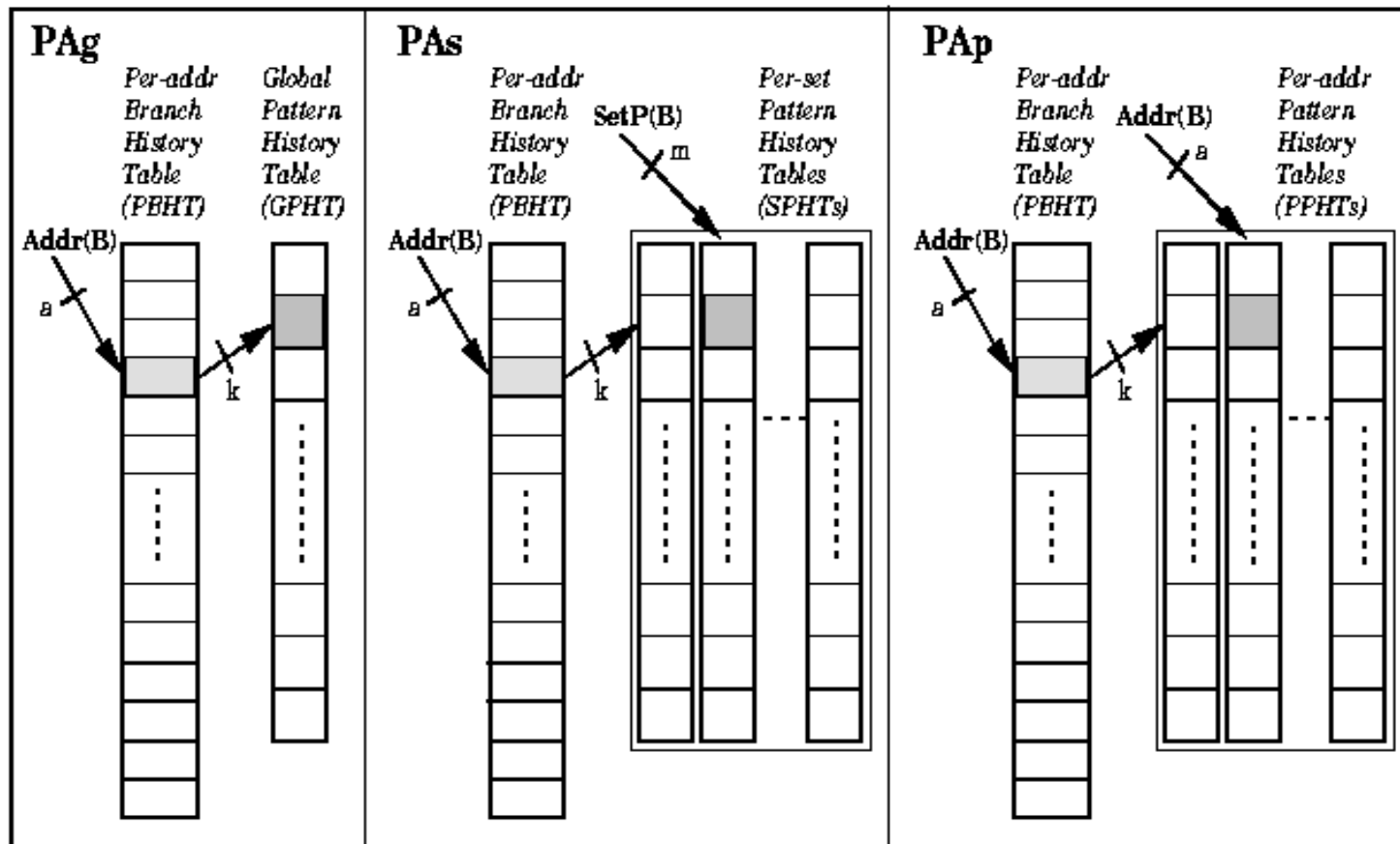
## The Global History Adaptive Schemes



## The Per-Set History Adaptive Schemes



## The Per-Address History Adaptive Schemes



# Other Two-Level Schemes

---

## Other Two-Level Schemes Exist

### **gshare**

- A single global branch history.
- A single branch history table.
- Index into branch history table is computed as the XOR of the branch address and the global branch history.

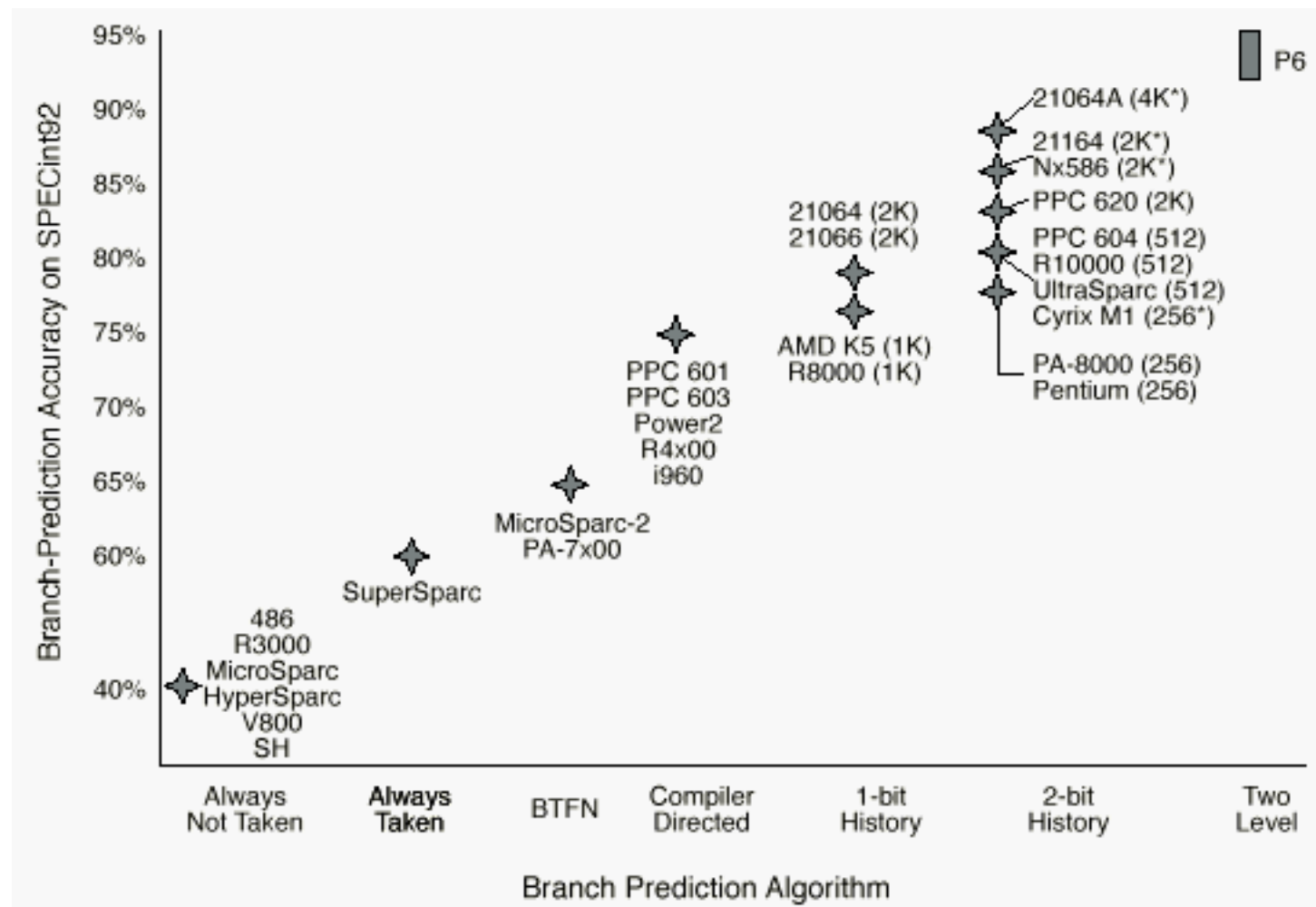
### **gselect**

- Similar to gshare.
- Index into branch history table is computed as the CONCATENATION of the branch address and the global branch history.

### **others**

- yags - yet another gshare
- variable history length predictors

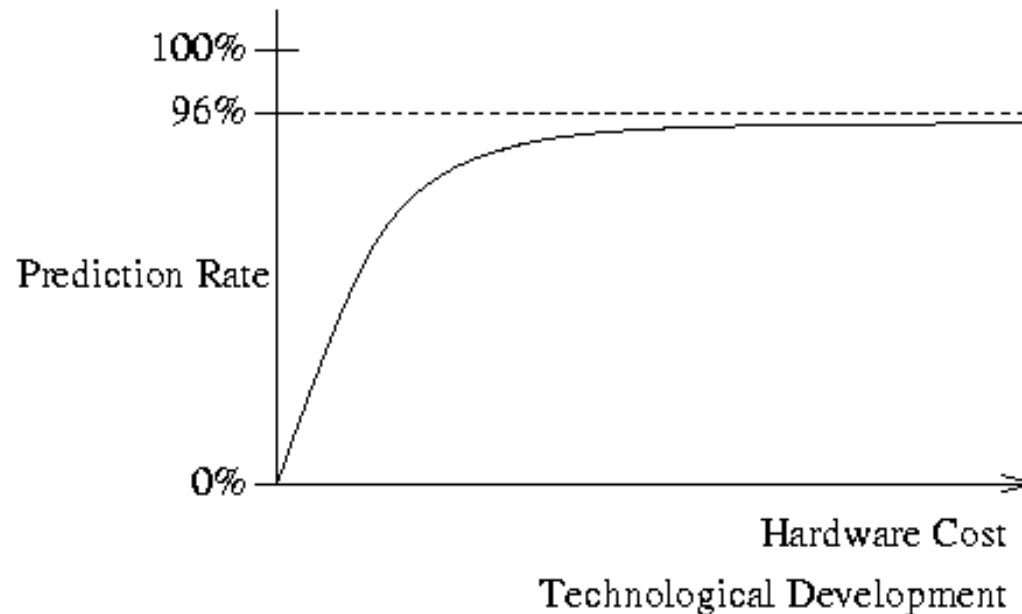
# Branch Prediction Comparisons



Source: Microprocessor Report

# Limits of Hardware Dynamic Branch Prediction

## Information Theoretic Viewpoint



- Consider branch prediction history.
- This branch history contains truly independent events.
- You can't predict the next white noise sample using knowledge gained from previous samples.
- Branch prediction using branch history has a theoretic limit.
- Trends indicate most prediction information extracted.



## Conventional Branch Prediction Techniques

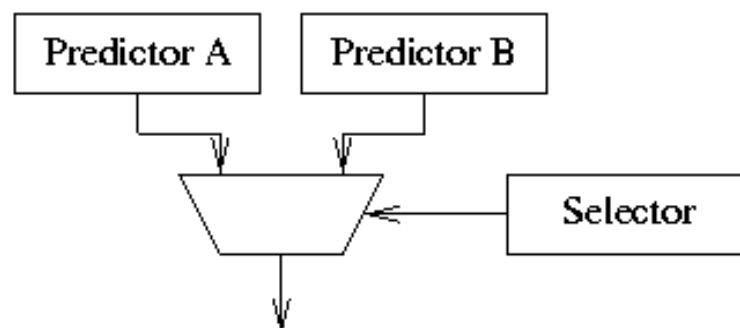
	Hardware	Compiler
Static	Uninteresting	Prediction bit, BTFNT
Dynamic	2-bit Counter, 2-Level	Unexplored, Focus of this Work

- Compiler/Static branch prediction
  - Compiler controlled - compile time decision
  - (+) No hardware limits - every branch has a real prediction
  - (+) Low cost
  - (-) Prediction cannot vary during program execution
  - (-) Performs poorly for unbiased branches
- Hardware/Dynamic branch prediction
  - Hardware controlled - state machine makes prediction
  - (+) Accuracy - varies during run-time
  - (-) Hardware has diminishing returns
  - (-) Area, power

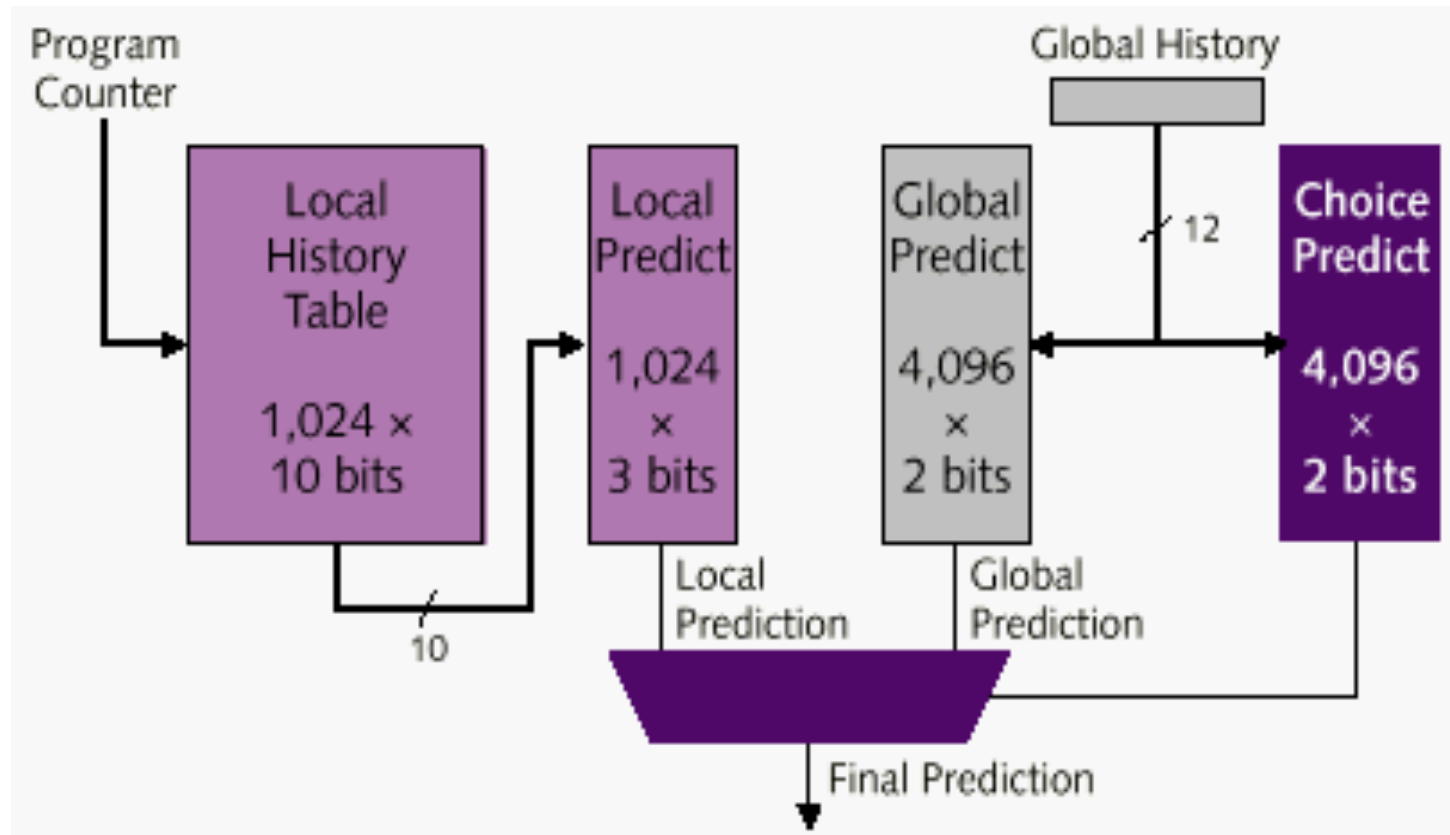
# Hybrid/Tournament Predictors

---

- If one is good, why not two, three, etc.?
- Hybrid (or multiple-scheme) branch prediction was proposed by Scott McFarling, WRL 1993
- Different predictors are adept at capturing different branch correlation. (global vs. local history)
- Use compiler or hardware to select predictor at run-time.
- Predictor selection mechanisms are very similar to branch predictors - predict which predictor is going to predict the branch correctly.

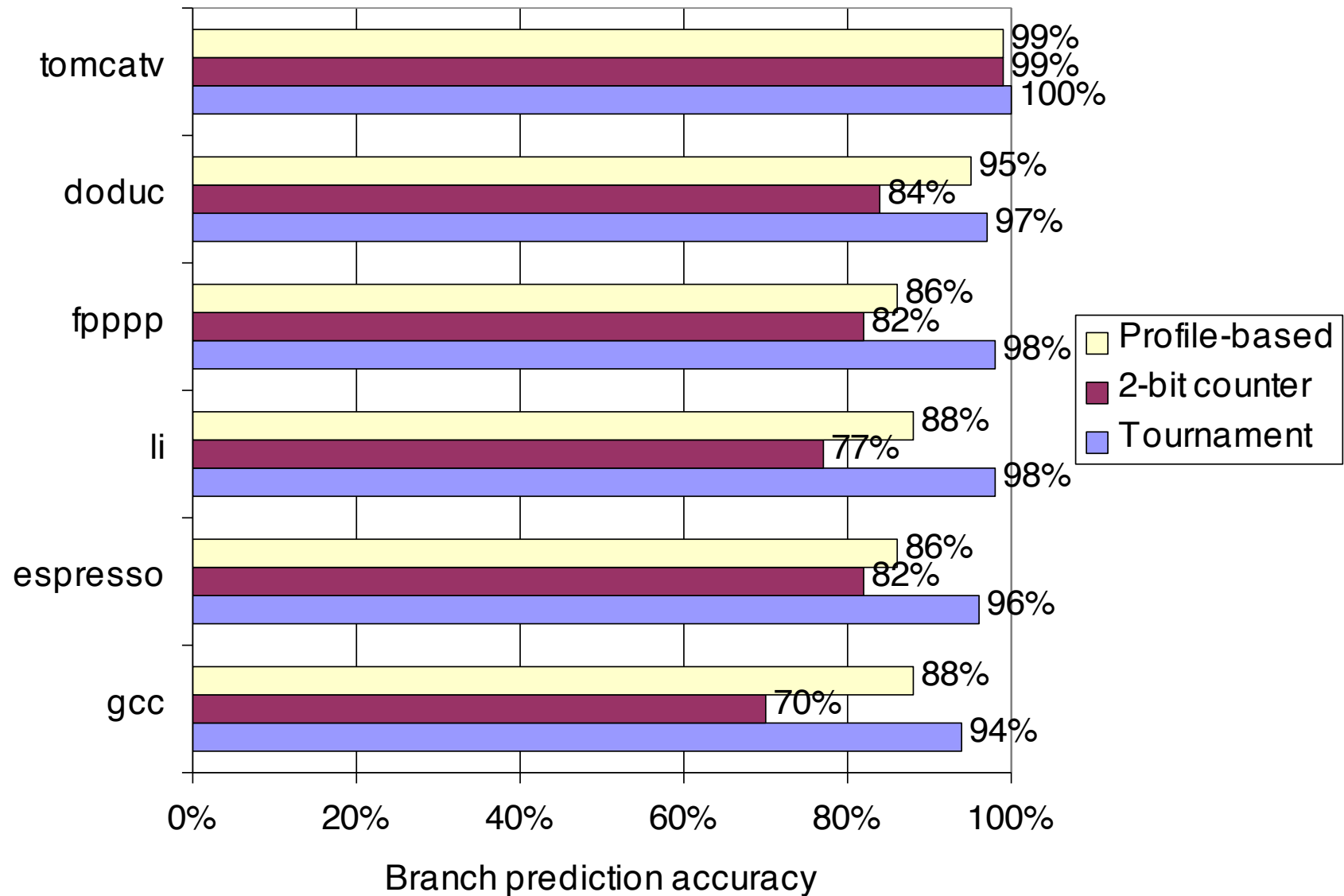


# 21264 Branch Prediction Logic

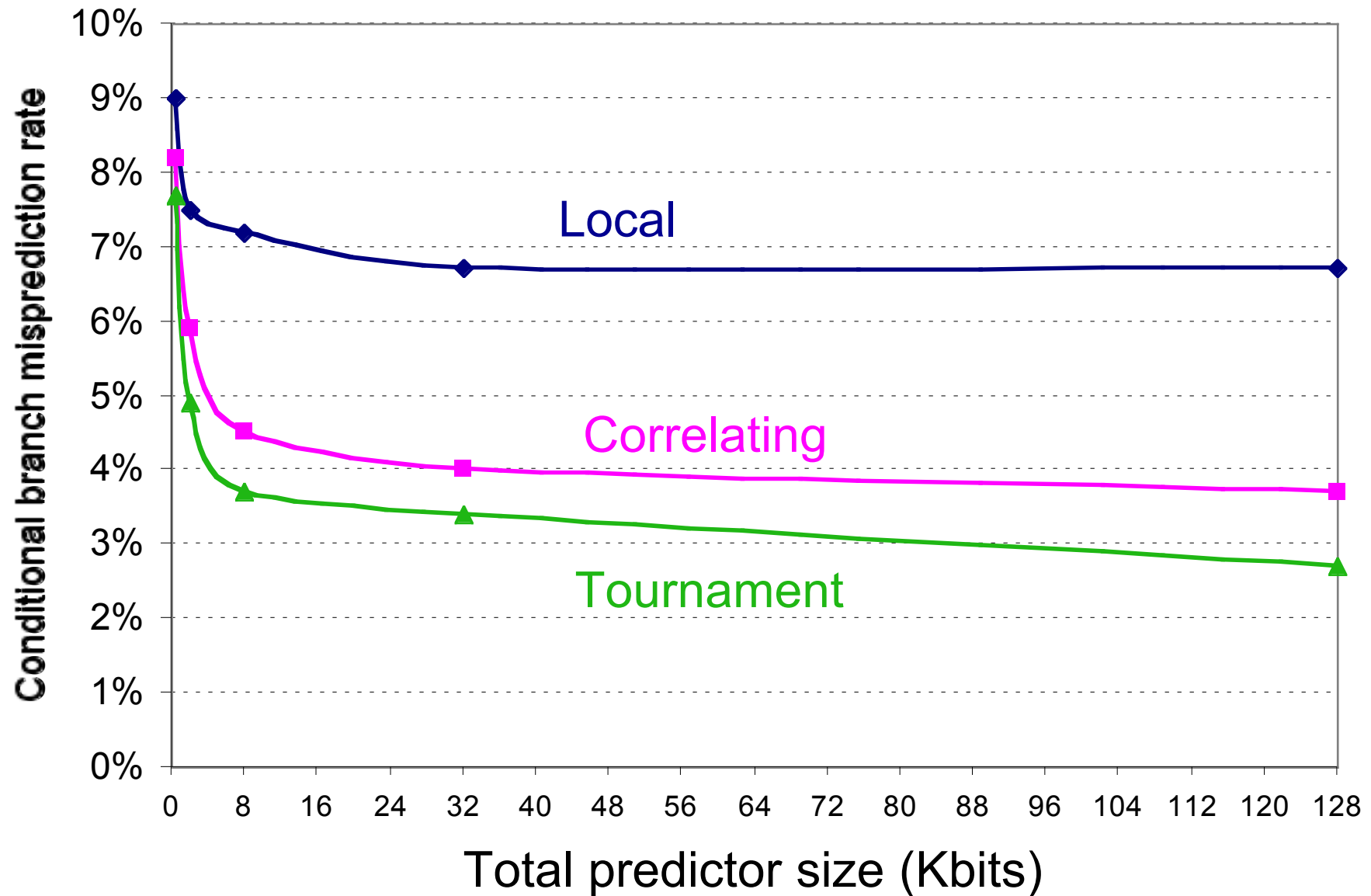


- 35Kb of prediction information
- 2% of total die size

# Accuracy of Branch Prediction



## Accuracy v. Size (SPEC89)



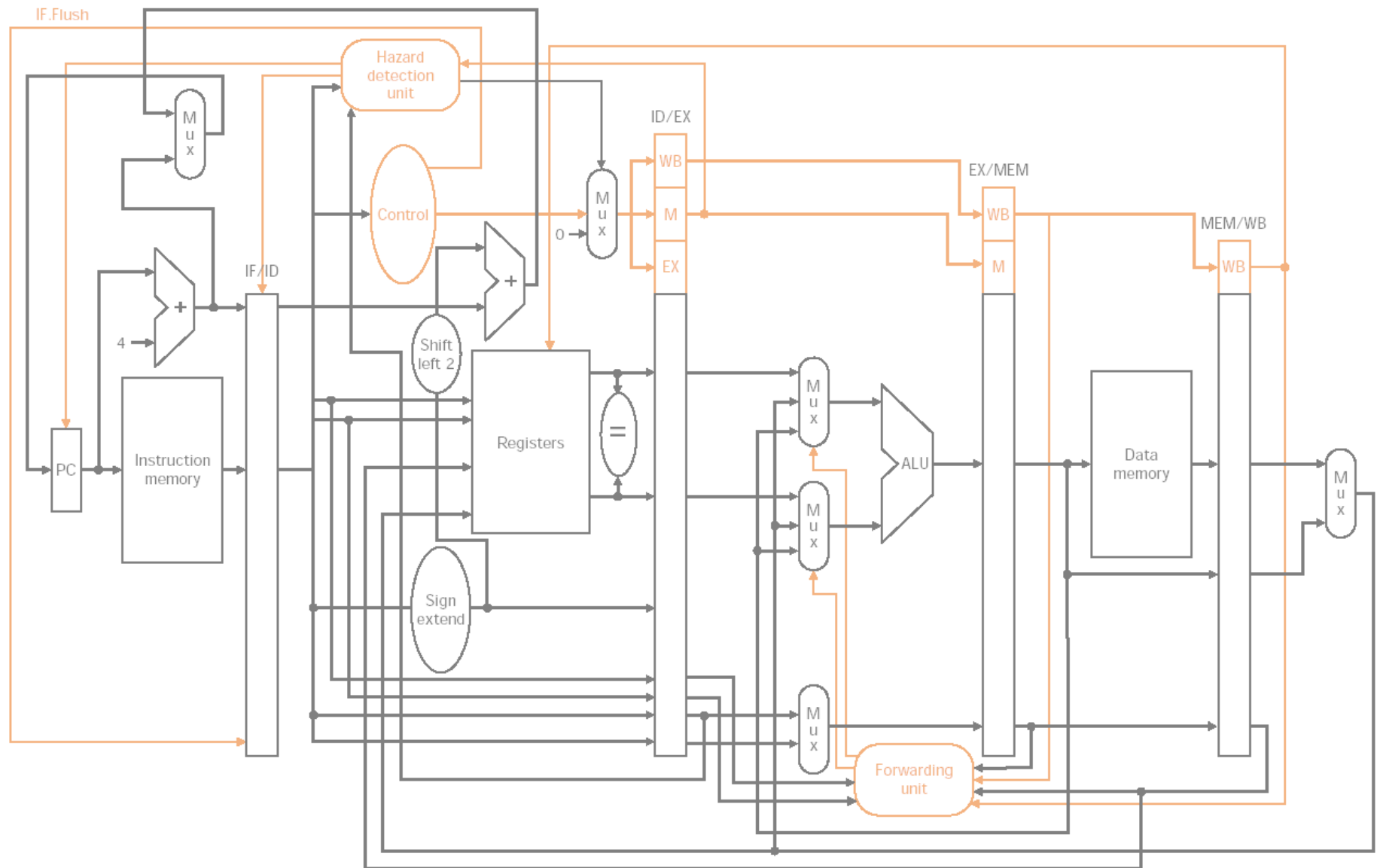
# FLASHBACK: Reduce Branch Delay

---

1. Move branch address calculation to decode stage (from MEM stage)
2. Move branch decision up (Harder
  - Bitwise-XOR, test for zero
  - Only need Equality testing
  - Much faster: No carry

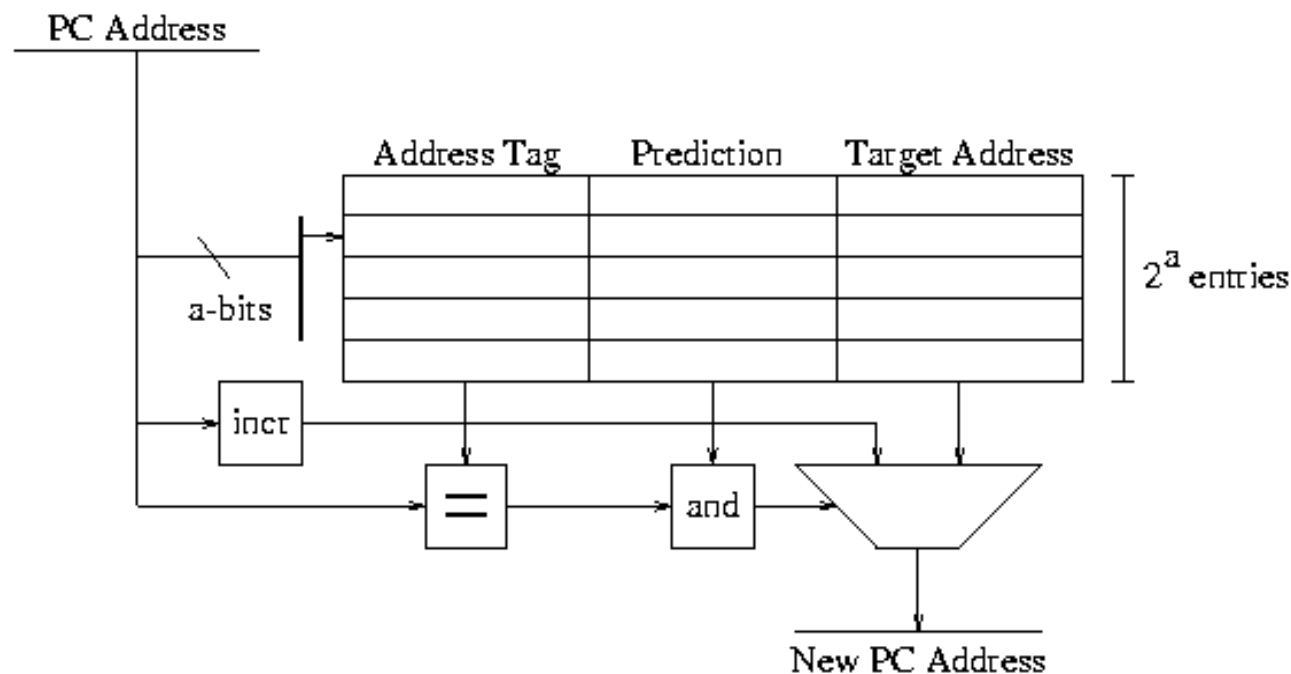
Everything is done in decode stage!!

# FLASHBACK: Reduce Branch Delay



# The Branch Target Buffer

- The Branch Target Buffer (BTB) is used by the fetch unit to determine the next Program Counter (PC) value.
- Some BTBs are tag-less to reduce table size and speed prediction delivery.



**First time a branch is encountered?**



## Another way to deal with branches

---

### Eliminate them!!!

- Predication (Intel IA-64/Itanium Processor and others)
- Predication vs. Prediction

# Summary

---

- Branches have personality
- Compiler/HW, Static/Dynamic
- Key ideas:
  - Correlation
  - Prediction accuracy
  - Hardware cost
  - Warm-up
- Hybrid Predictors
- Branch Target Buffers
- Predication