

Portland State University  
ECE 587/687

## Branch Prediction

© Copyright by Alaa Alameldeen and Haitham Akkary 2009

## Branch Penalty

- Example: Comparing perfect branch prediction to 90%, 95%, 99% prediction accuracy, and to no branch prediction
  - ◆ Processor has a 20-stage 6-wide pipeline, incorrectly predicted branch leads to pipeline flush
  - ◆ Program can have an average of 4 instructions retire per cycle, has 100,000 conditional branches out of 1 million instructions
  - ◆ Perfect BP:  $IPC = 1,000,000/250,000 = 4.00$
  - ◆ 90% BP accuracy: 1/10 branches incorrectly predicted
    - $IPC = 1,000,000/(250,000 + 0.1 \times 100,000 \times 20) = 2.22$  (80% slower)
  - ◆ 95% BP accuracy: 1/20 branches incorrectly predicted
    - $IPC = 1,000,000/(250,000 + 0.05 \times 100,000 \times 20) = 2.85$  (40% slower)
  - ◆ 99% BP accuracy: 1/100 branches incorrectly predicted
    - $IPC = 1,000,000/(250,000 + 0.01 \times 100,000 \times 20) = 3.70$  (8% slower)
  - ◆ No BP: Fetch stalled until branch is resolved (5 pipeline stages)
    - $IPC = 1,000,000/(250,000 + 100,000 \times 5) = 1.33$  (300% slower – 3x)

Portland State University – ECE 587/687 – Spring 2009 2

## Reducing Branch Costs with Dynamic Hardware Prediction

- Branch prediction basics:
  - ◆ We need to predict conditional branch outcome to select the address for next instruction fetch
    - PC + 4
    - Or branch target address
  - ◆ Also we need to quickly determine the branch target address
    - Direct branches
    - Register indirect branches
    - Returns

Portland State University – ECE 587/687 – Spring 2009 3

## Predicting Conditional Branch Outcomes

- Simplest dynamic branch prediction scheme uses a *branch-prediction buffer* or *branch history table*
  - ◆ Small memory indexed by the lower portion of the branch address
  - ◆ Stores previous branch outcomes to predict next outcome
  - ◆ Memory is not *tagged*
  - ◆ Prediction may have been put in the entry by a different branch (Aliasing)

PC

PC [11:2]

1K entries prediction buffer

Portland State University – ECE 587/687 – Spring 2009 4

## Predicting Conditional Branch Outcomes

- 1-bit prediction buffer stores the last executed branch outcome, and uses it to predict the next outcome
  - ◆ If bit = 1, branch is predicted taken
  - ◆ If bit = 0, branch is predicted not-taken
- A simple 1-bit scheme has a performance shortcoming
  - ◆ A series of branch outcomes and corresponding predictions :
 

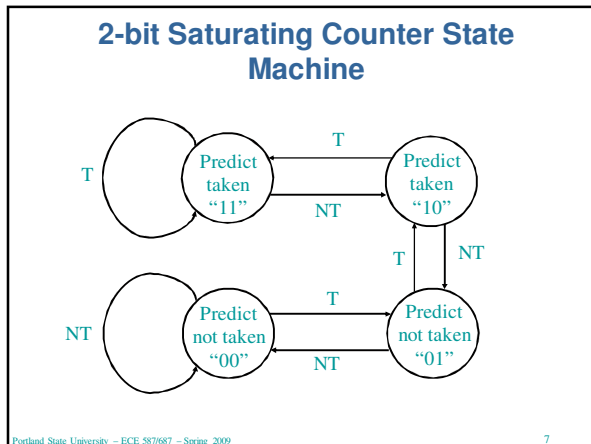
outcomes	1111011110111101
predictions	1111011110111110
mispredictions	111 <u>1</u> 0111 <u>1</u> 011 <u>1</u> 0

Portland State University – ECE 587/687 – Spring 2009 5

## Predicting Conditional Branch Outcomes

- 2-bit saturating counter often used
  - ◆ Branch taken ==> increment state
    - Max state "11" stays at "11" when incremented
  - ◆ Branch not-taken ==> decrement state
    - Min state "00" stays at "00" when decremented
  - ◆ "11" and "10" are predict taken states
  - ◆ "00" and "01" are predict not-taken states

Portland State University – ECE 587/687 – Spring 2009 6



### Predicting Conditional Branch Outcomes

- Assuming initial state to be “11”, branch outcomes and corresponding predictions now look as follows:

outcomes	1111011110111101
states	333323333233332
predictions	1111111111111111
mispredictions	111 <u>1</u> 111 <u>1</u> 1111 <u>1</u> 1

Portland State University – ECE 587/687 – Spring 2009 8

### Correlating Branch Predictors

- 2-bit prediction schemes use the recent behavior of a single branch to predict the future behavior of that branch
- Behavior of longer sequence of branch execution *history* often provides more accurate prediction outcome
- Behavior of *other* branches rather than just the branch we are trying to predict is sometimes important
  - ◆ Because outcomes of different branches often correlate
  - ◆ Global branch history
- For some branches, prior history execution of the branch is important
  - ◆ Because of loops
  - ◆ Local branch history

Portland State University – ECE 587/687 – Spring 2009 9

### Two-Level Adaptive Branch Prediction

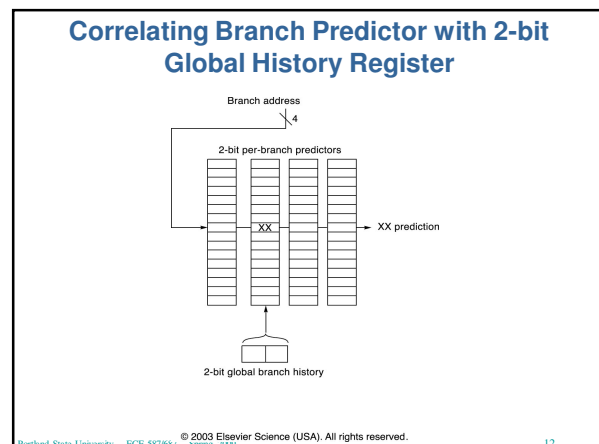
- Two main structures
  - ◆ Branch History Register (BHR) or Branch History Table (BHT)
  - ◆ Pattern History Table (PHT)
  - ◆ Basic structure of the branch predictor: Yeh&Patt Figure 1
  - ◆ Updating predictions using automaton: Yeh&Patt Figure 2
- Three different flavors
  - ◆ Global History Register and Global Pattern History Table (GAg)
  - ◆ Per-address Branch History Table and Global Pattern History Table (PAG)
  - ◆ Per-address Branch History Table and Per-address Pattern History Tables (PAP)

Portland State University – ECE 587/687 – Spring 2009 10

### Correlating Branch Predictors: Code Example

if (aa == 2)	DSUBUI R3,R1,#2
aa = 0;	BNEZ R3,L1
DADD R1,R0,R0	
if (bb == 2)	L1: DSUBUI R3,R2,#2
bb = 0;	BNEZ R3,L2
DADD R2,R0,R0	
if (aa != bb) {	L2: DSUBU R3,R1,R2
	BEQZ R3,L3

Portland State University – ECE 587/687 – Spring 2009 11



### Two-Level Adaptive Branch Prediction: Discussion

- Cost-effectiveness of three flavors
  - ◆ GAg has too much branch interference, needs long history
  - ◆ PAp needs lots of space for Per-address PHT
  - ◆ PAg is the most cost-effective
- Context switch
  - ◆ GAg almost unaffected
  - ◆ PAg, PAp degraded
  - ◆ Pros and cons for saving branch history on a context switch?

Portland State University – ECE 587/687 – Spring 2009 13

### Other Branch Prediction Strategies

- McFarling's Paper:
  - ◆ Bimodal Predictor: Figure 1
  - ◆ PAg and GAg: Figure 4 and 6
  - ◆ Global Predictor with Index Selection: Figure 8
  - ◆ Global History with Index Sharing (GShare): Figure 10
- Using perceptrons instead of 2-bit saturating counters
  - ◆ Jimenez&Lin's paper (skim, not in exam)
  - ◆ Provides higher prediction accuracy

Portland State University – ECE 587/687 – Spring 2009 14

### Adaptively Combining Branch Predictors

- Some branches are predicted more accurately with *global* predictors
- Other branches are predicted better with *local* predictors
- It is possible to combine both types of predictors, and dynamically select the right predict for the right branch
- The selector is yet another predictor with 2-bit state machine per entry

Portland State University – ECE 587/687 – Spring 2009 15

### Branch Target Buffer (BTB)

- A cache that stores branch targets
- Accessed by the address of the instruction currently fetched
- Allows branch target to be read in the IF stage
  - ◆ When a branch is predicted taken, the fetch of the instruction at the branch target address can proceed immediately in the next cycle
  - ◆ Stall cycles that would have been needed to wait for the decoding of the branch and the computation of the target are saved

Portland State University – ECE 587/687 – Spring 2009 16

### Branch target buffer

© 2003 Elsevier Science (USA). All rights reserved.  
Portland State University – ECE 587/687 – Spring 2009 17

### Predicting return address using Return Address Stack (RAS)

- Indirect branches have multiple potential targets, since address comes from a register, which can have many possible values
- Branch target buffers could be used for indirect branch target prediction
  - ◆ However, many mispredictions can happen because the BTB can store only one target per branch
- Most indirect branches come from return instructions

Portland State University – ECE 587/687 – Spring 2009 18

### Return address stack

- A small address buffer organized as a stack
- When a Call is encountered the Return address (which is Call address + 4) is pushed onto the RAS
- When a Return instruction is encountered, the address from the top of the RAS is popped and used as the target

### Reading Assignment

- Eric Rotenberg et al., "Trace Cache: A Low Latency Approach to High Bandwidth Instruction Fetching," MICRO 1996 (Review)
- Daniel Jimenez and Calvin Lin, "Dynamic Branch Prediction with Perceptrons," HPCA 2001 (Skim)