

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Computer Architecture
ECE 568

Part 7

Dynamic Branch Prediction

Israel Koren
Fall 2009

ECE568/Koren Part.7 .1

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

Static Branch Prediction

- ◆ **Simplest: Predict taken**
 - average misprediction rate = untaken branch frequency, which for the SPEC programs is 34%
 - Unfortunately, the correct prediction rate ranges from not very accurate (41%) to highly accurate (91%)
- ◆ **Predict on the basis of branch direction?**
 - choosing backward-going branches to be taken (**loop**)
 - forward-going branches to be not taken (**if**)
 - SPEC programs, however, most forward-going branches are taken => predict taken is better
- ◆ **Predict branches on the basis of profile information collected from earlier runs**
 - Misprediction varies from 5% to 22%

ECE568/Koren Part.7 .2

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

Seven Branch Prediction Schemes

- ◆ 1-bit Branch-Prediction
- ◆ 2-bit Branch-Prediction
- ◆ Correlating Branch Prediction
- ◆ Tournament Branch Predictor
- ◆ Branch Target Buffer
- ◆ Conditionally Executed Instructions
- ◆ Return Address Predictors

- Branch Prediction even more important when N instructions per cycle are issued
- Amdahl's Law => relative impact of the control stalls will be larger with the lower potential CPI in an n -issue processor

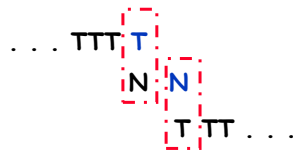
ECE568/Koren Part.7 .3

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

Dynamic Branch Prediction

- ◆ Performance = $f(\text{accuracy}, \text{cost of misprediction})$
- ◆ Branch History Table (BHT): Lower bits of PC address index table of 1-bit values
 - Says whether or not branch taken last time (T-Taken, N)
 - No full address check (saves HW, but may be wrong)
- ◆ Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 9 iterations before exit):
 - End of loop case, when it exits instead of looping as before
 - First time through loop on *next* time through code, when it predicts *exit* instead of looping
 - Only 77.8% accuracy if 9 iterations per loop on average



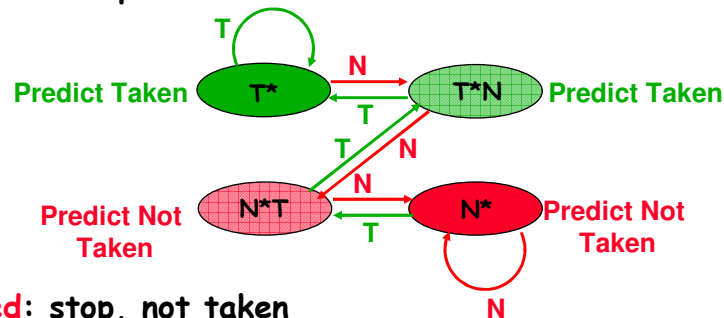
ECE568/Koren Part.7 .4

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

2-bit Branch Prediction - Scheme 1

- ◆ Better Solution: 2-bit scheme where change prediction only if get misprediction *twice* but return in one step:



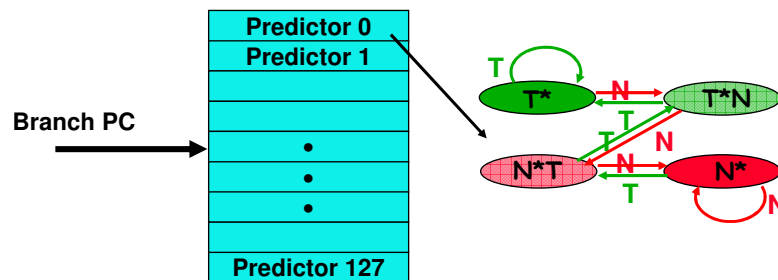
(Jim Smith, 1981)

ECE568/Koren Part.7 .5

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

Branch History Table (BHT)



- ◆ BHT is a table of "Predictors"
 - 2-bit, saturating counters indexed by PC address of Branch
- ◆ In Fetch phase of branch:
 - Predictor from BHT used to make prediction
- ◆ When branch completes:
 - Update corresponding Predictor

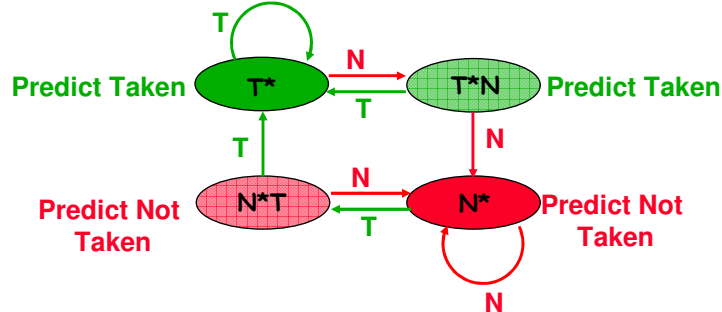
ECE568/Koren Part.7 .6

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

2-bit Branch Prediction - Scheme 2

- ◆ Another Solution: 2-bit scheme where change prediction (in either direction) only if get misprediction *twice* :



- ◆ **Red**: stop, not taken
- ◆ **Green**: go, taken

Lee & A. Smith, IEEE
Computer, Jan 1984

ECE568/Koren Part.7 .7

Copyright 2008 Koren UMass

Comparison

<p>Actual: T N T T T N T</p> <p>State: T* T* T*N T* T* T* T*N T*</p> <p>Predicted: T T T T T T T T</p>	}	For both schemes
<p>Actual: N N T N N T N N</p> <p>State: N* N* N* N*T N* N* N*T N*</p> <p>Predicted: N N N N N N N N</p>		
<p>Actual: N N T T N N T T</p> <p>State: N* N* N* N*T T*N N*T N* N*T</p> <p>Predicted: N N N N T N N N</p>		Scheme 1
<p>Actual: N N T T N N T T</p> <p>State: N* N* N* N*T T* T*N N* N*T</p> <p>Predicted: N N N N ? ? ? ?</p>		Scheme 2

ECE568/Koren Part.7 .8

Copyright 2008 Koren UMass

Further Comparison

②

①

- ◆ Alternating taken / not-taken
- ◆ Your worst-case prediction scenario
- ◆ Both schemes achieve 80-95% accuracy with only a small difference in behavior

ECE568/Koren Part.7 .9 Copyright 2008 Koren UMass

Correlating Branches

Idea: taken/not taken of recently executed branches is related to behavior of present branch (as well as the history of that branch behavior)

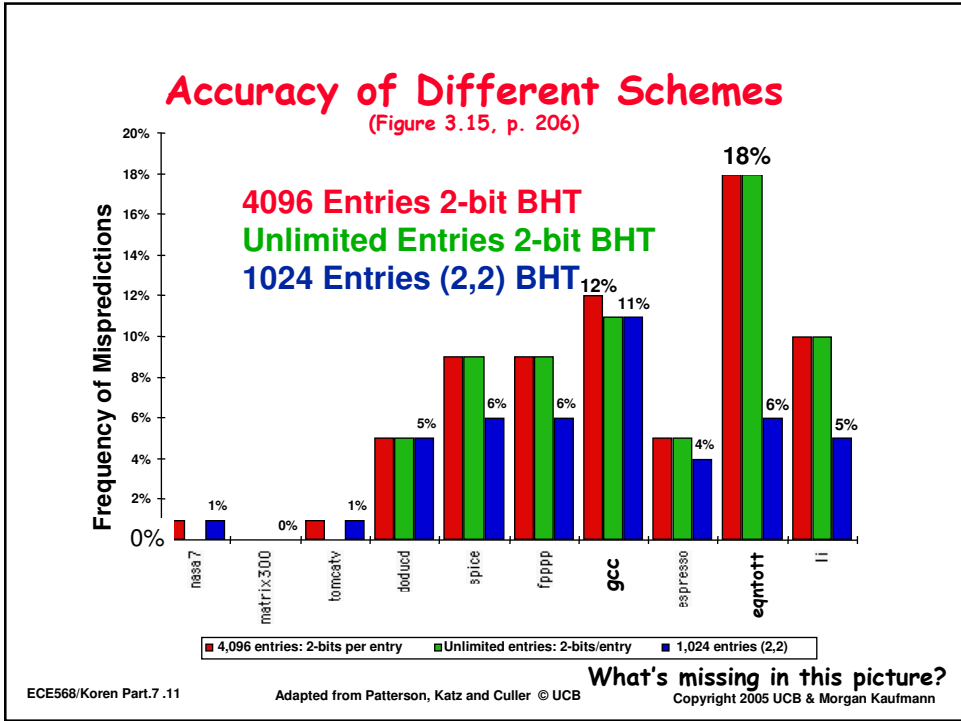
- Then behavior of recent branches selects between, say, 4 predictions of next branch, updating just that prediction

◆ (2,2) predictor: 2-bit global, 2-bit local

Branch address (4 bits)

2-bit recent global branch history
(01 = not taken then taken)

ECE568/Koren Part.7 .10 Adapted from Patterson, Katz and Culler © UCB Copyright 2005 UCB & Morgan Kaufmann



Re-evaluating Correlation

- ◆ Several SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:

program	branch %	static	# = 90%
compress	14%	236	13
eqntott	25%	494	5
gcc	15%	9531	2020
mpeg	10%	5598	532
real gcc	13%	17361	3214

- ◆ Real programs + OS more like gcc
- ◆ Small benefits of correlation beyond benchmarks?
- ◆ Mispredict because either:
 - Wrong guess for that branch
 - Got branch history of wrong branch when indexing the table
- ◆ For SPEC92, 4096 about as good as infinite table
 - Misprediction mostly due to wrong prediction
- ◆ Can we improve using global history?

ECE568/Koren Part.7 .12 Adapted from Patterson, Katz and Culler © UCB Copyright 2005 UCB & Morgan Kaufmann

Tournament Predictors

- ◆ Motivation for correlating branch predictors: 2-bit local predictor failed on important branches; by adding global information, performance improved
- ◆ Tournament predictors: use two predictors, 1 based on global information and 1 based on local information, and combine with a selector
- ◆ Hopes to select right predictor for right branch (or right context of branch)

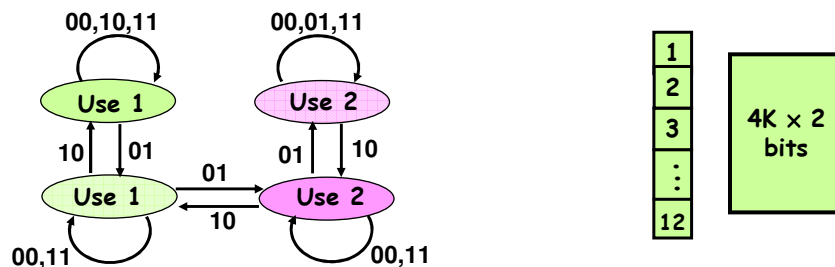
ECE568/Koren Part.7 .13

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

Tournament Predictor in Alpha 21264

- ◆ 4K 2-bit counters to choose from among a global predictor and a local predictor
- ◆ **Global predictor** also has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
 - 12-bit pattern: i th bit is 0 => i th prior branch not taken; i th bit is 1 => i th prior branch taken;



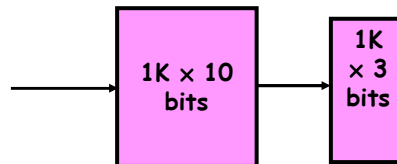
ECE568/Koren Part.7 .14

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

Tournament Predictor in Alpha 21264

- ◆ **Local predictor** consists of a 2-level predictor:
 - **Top level** a local history table consisting of 1024 10-bit entries; each 10-bit entry corresponds to the most recent 10 branch outcomes for the entry. 10-bit history allows patterns 10 branches to be discovered and predicted
 - **Next level** Selected entry from the local history table is used to index a table of 1K entries consisting a 3-bit saturating counters, which provide the local prediction
- ◆ **Total size: $4K \times 2 + 4K \times 2 + 1K \times 10 + 1K \times 3 = 29K$ bits!**
(~180K transistors)

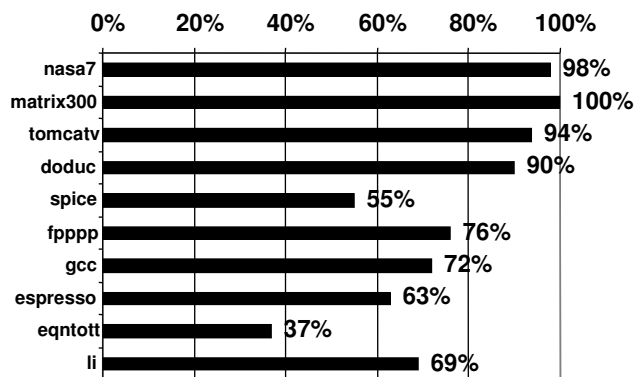


ECE568/Koren Part.7 .15

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

% of predictions from local predictor in Tournament Prediction Scheme



ECE568/Koren Part.7 .16

Adapted from Patterson, Katz and Culler © UCB

Copyright 2005 UCB & Morgan Kaufmann

Accuracy of Branch Prediction

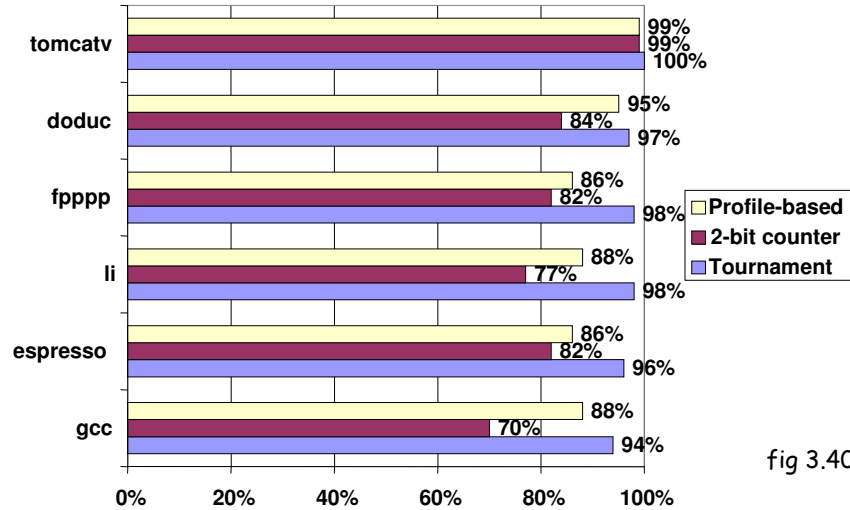
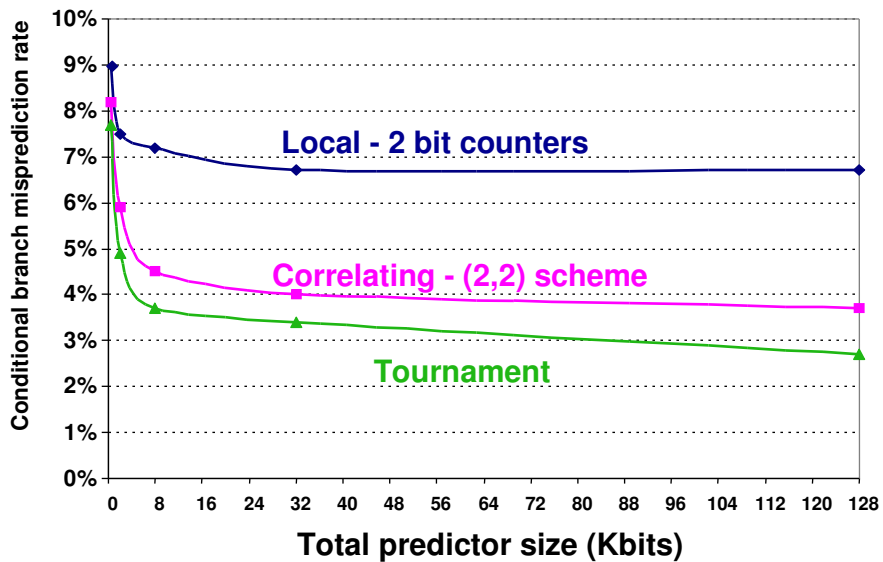


fig 3.40

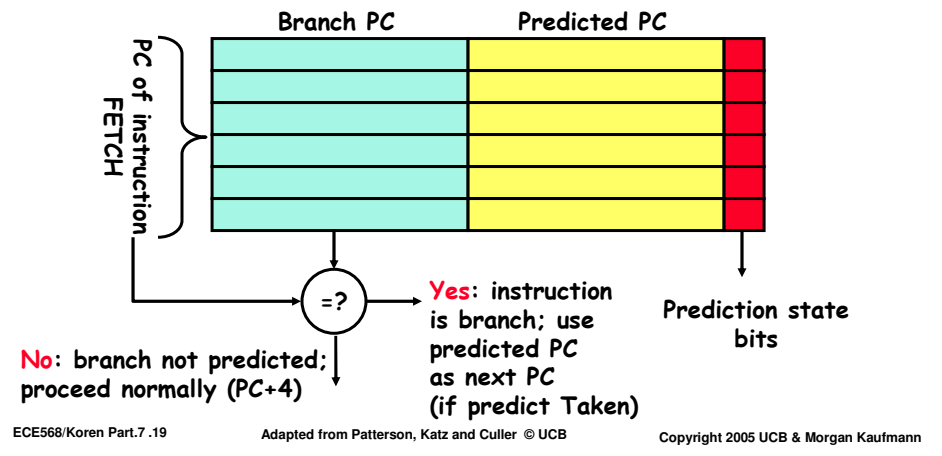
◆ Profile: branch profile from last execution (static in that is encoded in instruction, but profile)

Accuracy v. Size (SPEC89)



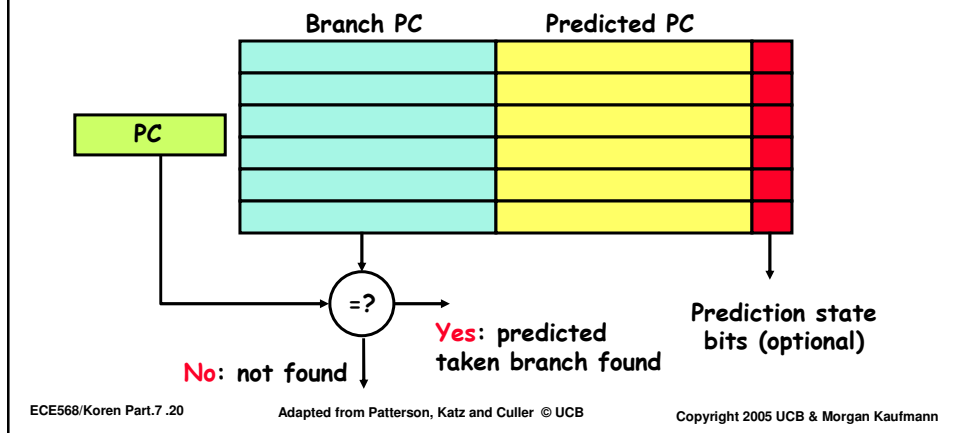
Need Address at Same Time as Prediction

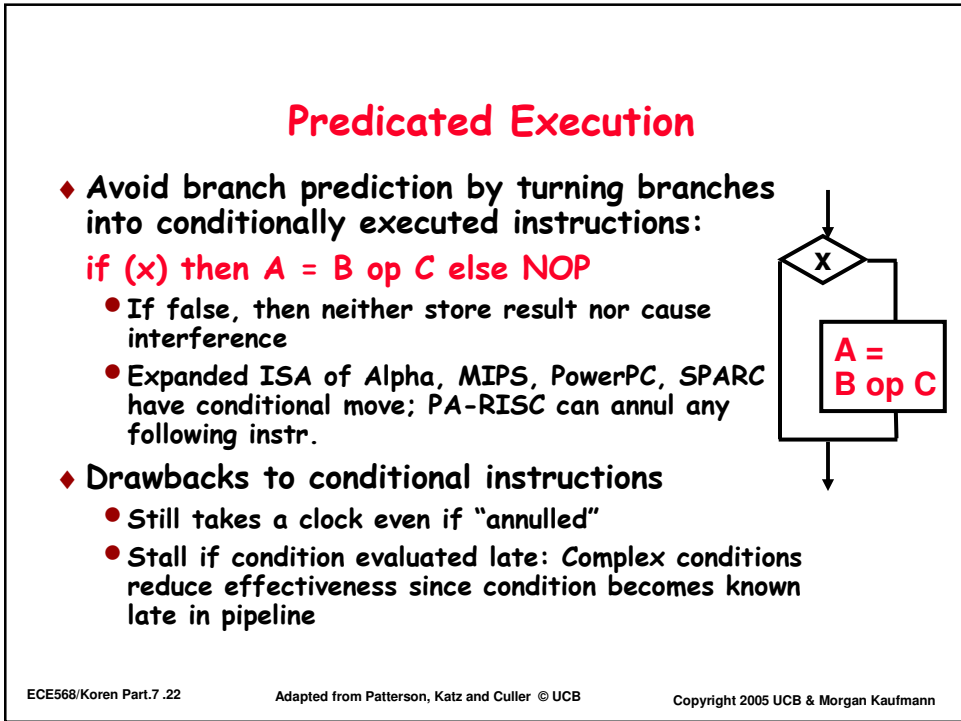
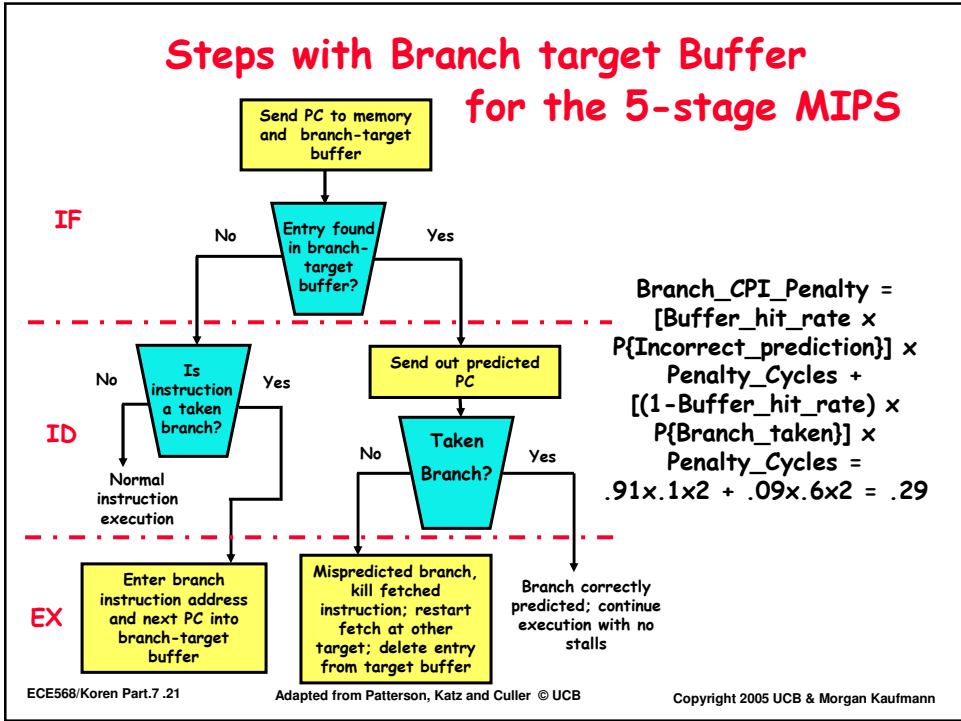
- ◆ Branch Target Buffer (BTB): Address of branch used as index to get prediction AND branch address (if taken)
 - Note: must check for branch match now, since can't use wrong branch address (Figure 3.19, 3.20)



Branch Target "Cache"

- ◆ Branch Target cache - Only predicted taken branches
- ◆ "Cache" - Content Addressable Memory (CAM) or Associative Memory (see figure)
- ◆ Use a big Branch History Table & a small Branch Target Cache





Special Case: Return Addresses

- ◆ Register Indirect branch - hard to predict address
- ◆ SPEC89 85% such branches for procedure return
- ◆ Since stack discipline for procedures, save return address in small buffer that acts like a stack: 8 to 16 entries has small miss rate

Pitfall: Sometimes dumber is better

- ◆ Alpha 21264 uses tournament predictor (29 Kbits)
- ◆ Earlier 21164 uses a simple 2-bit predictor with 2K entries (or a total of 4 Kbits)
- ◆ SPEC95 benchmarks, 21264 outperforms
 - 21264 avg. 11.5 mispredictions per 1000 instructions
 - 21164 avg. 16.5 mispredictions per 1000 instructions
- ◆ Reversed for transaction processing (TP) !
 - 21264 avg. 17 mispredictions per 1000 instructions
 - 21164 avg. 15 mispredictions per 1000 instructions
- ◆ TP code much larger & 21164 hold 2X branch predictions based on local behavior (2K vs. 1K local predictor in the 21264)