

# Branch Prediction

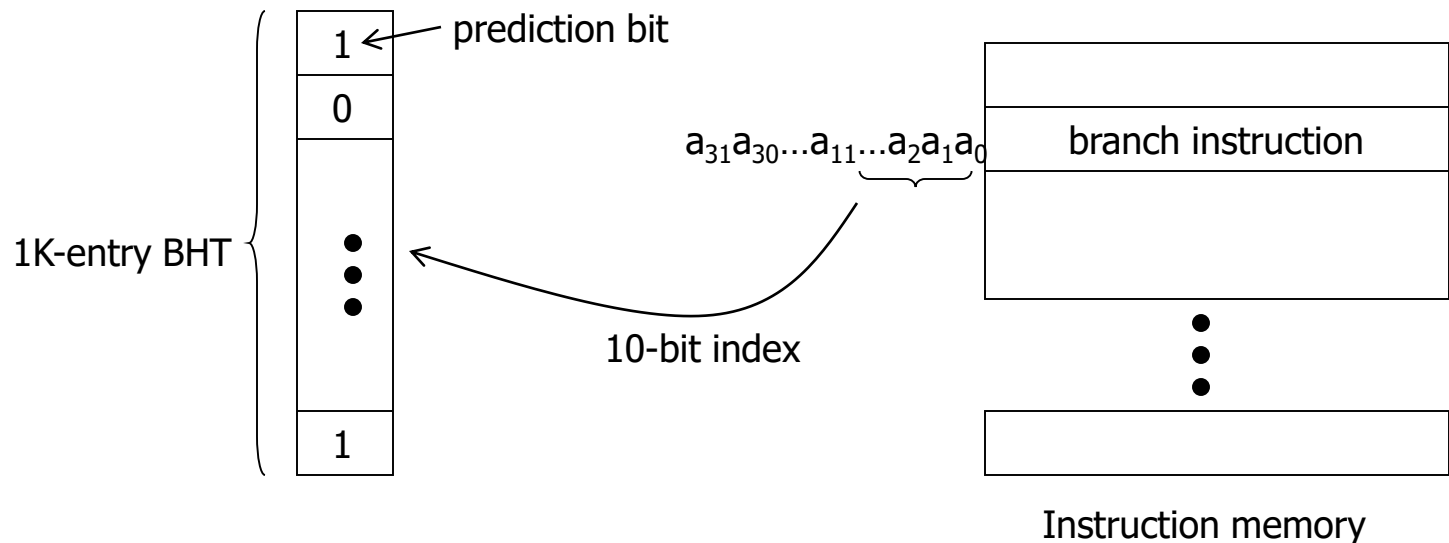
# Static Prediction

- Does not take into account the *run-time history* of the particular branch instruction – whether it was taken or not taken recently, how often it was taken or not taken, etc.
- Simplest static prediction:
  - predict *always taken*
  - predict *always not taken*
- More complex static prediction:
  - performed at compile time by analyzing the program...

# Dynamic Hardware Branch Prediction

# Dynamic prediction: 1-bit Predictor

- *Branch- prediction buffer or branch history table (BHT)* is a cache indexed by a fixed lower portion of the address of the branch instruction
- *1-bit prediction*: for each index the BHT contains one *prediction bit* (also called *history bit*) that says if the branch was last taken or not – *prediction is that branch will do the same again*

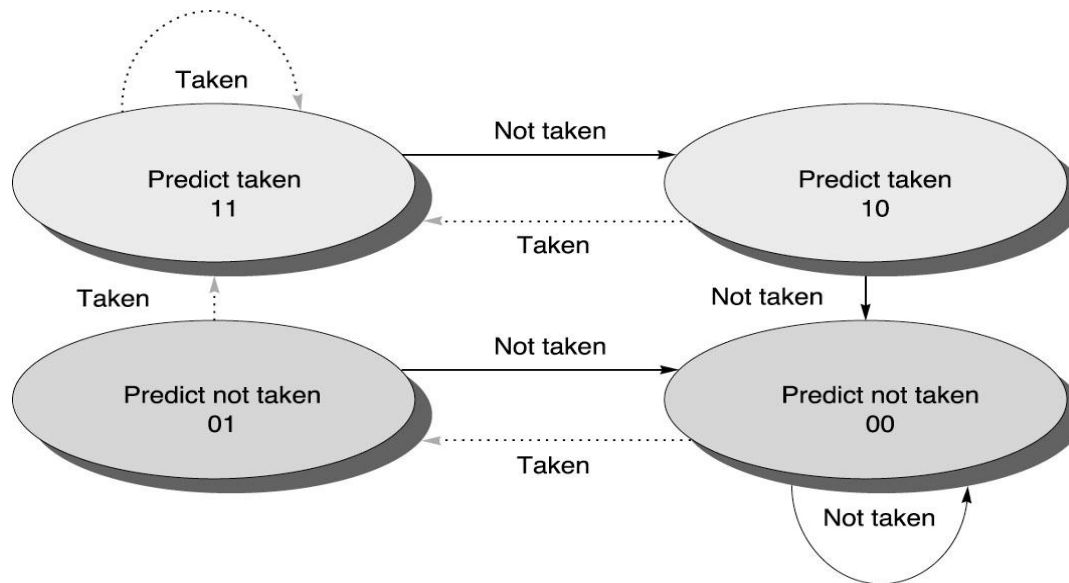


# Dynamic prediction: 1-bit Predictor

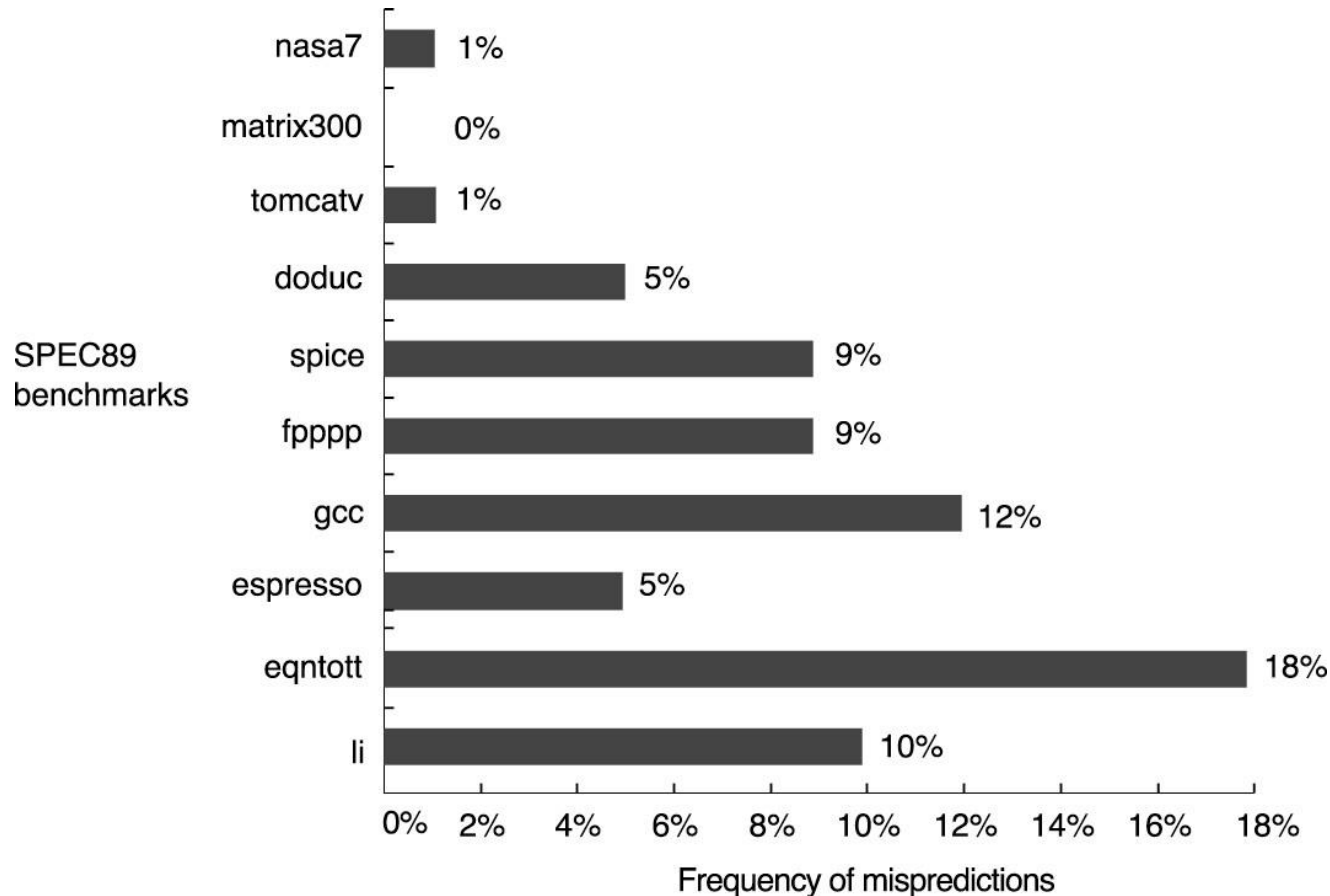
- Meaning of prediction bit
  - 1 = branch was last taken
  - 0 = branch was last not taken
- Using the BHT
  - index into the BHT and use the prediction bit to predict branch behavior
    - note the prediction bit may have been set by a different branch instruction with the same lower address bits but that does not matter – the history bit is simply a *hint*
  - if prediction is wrong, invert prediction bit
- Example: *Consider a loop branch that is taken 9 times in a row and then not taken once. What is the prediction accuracy of 1-bit predictor for this branch assuming only this branch ever changes its corresponding prediction bit?*
  - Answer: 80%. Because there are two mispredictions – one on the first iteration and one on the last iteration. *Why?*

# Dynamic prediction: 2-bit Predictor

- *2-bit prediction*: for each index the BHT contains two prediction bits that change as in the figure below
- Key idea: the prediction must be wrong *twice* for it to be changed
- Example: *What is the prediction accuracy of a 2-bit predictor on the loop of the previous example?*

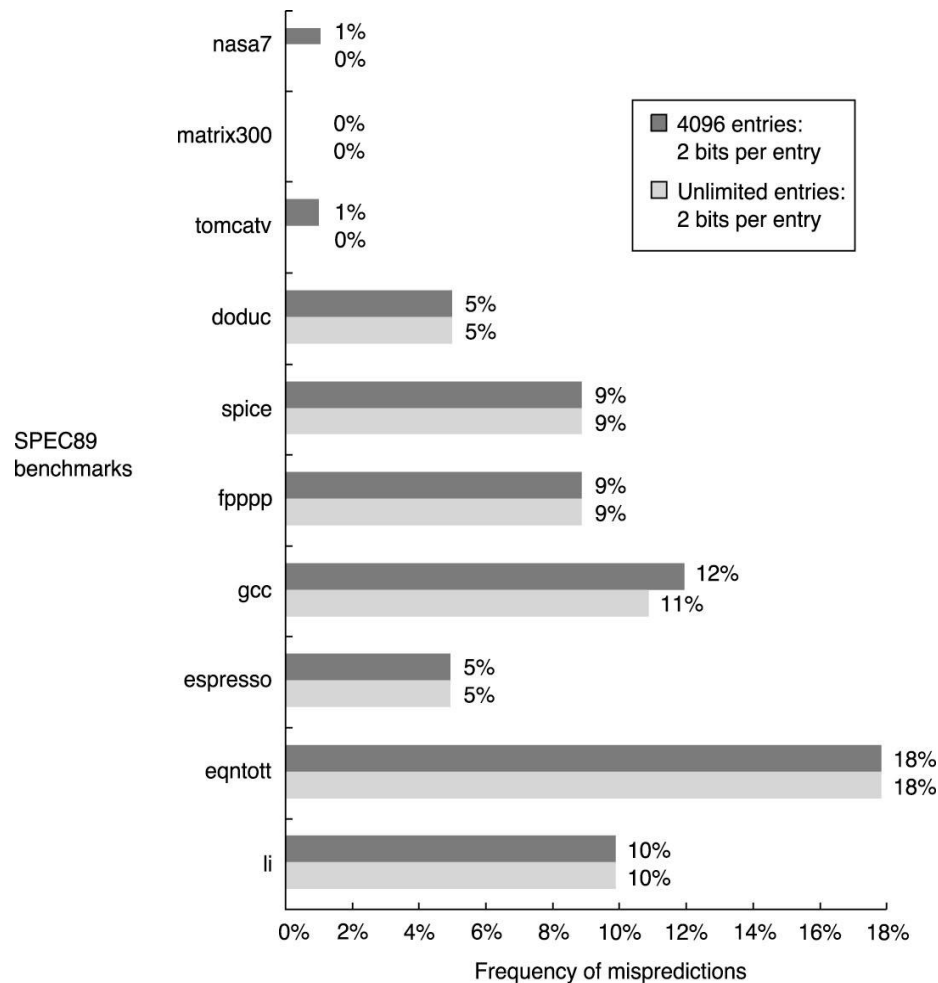


# 2-bit Predictor Statistics



**Prediction accuracy of 4K-entry 2-bit prediction buffer on SPEC89 benchmarks: accuracy is lower for integer programs (gcc, espresso, eqntott, li) than for FP**

# 2-bit Predictor Statistics



**Prediction accuracy of 4K-entry 2-bit prediction buffer vs. "infinite" 2-bit buffer:**  
**increasing buffer size from 4K does not significantly improve performance**



# n-bit Predictors

- Use an n-bit counter which, therefore, represents a value X where  $0 \leq X \leq 2^n - 1$ 
  - increment X if branch is taken (to a max of  $2^n$ )
  - decrement X if branch is not taken (to a min of 0)
  - If  $X \geq 2^{n-1}$ , then predict taken; otherwise, untaken
- Studies show that there is no significant improvement in performance using n-bit predictors with  $n > 2$ , so *2-bit predictors are implemented in most systems*

# **Correlating Predictors**

# Correlating Predictors

if (aa == 2)		DSUBUI	R3, R1, #2	
aa = 0;		BNEZ	R3, L1	; <b>branch b1</b> (aa != 2)
if (bb == 2)		DADD	R1, R0, R0	; aa = 0
bb=0;	L1:	DSUBUI	R3, R2, #2	
if (aa != bb) {...		BNEZ	R3, L2	; <b>branch b2</b> (bb != 2)
		DADD	R2, R0, R0	; bb = 0
	L2:	DSUB	R3, R2, R1	; R3 = aa - bb
		BEQZ	R3, L3	; <b>branch b3</b> (aa == bb)

**Code fragment from  
eqntott SPEC89 benchmark**

**Corresponding MIPS code:  
aa is in R1, bb is in R2**

- Key idea: branch b3 behavior is correlated with the behavior of branches b1 and b2
  - because *if branches b1 and b2 are both not taken*, then the statements following the branches will set aa=0 and bb=0  
 $\Rightarrow$  *b3 will be taken*

# Correlating Predictors: Simple Example

```

if (d == 0)           BNEZ      R1, L1      ; branch b1   (d != 0)
    d = 1;           DADDIU    R1, R0, #1   ; d==0, so d=1
if (d == 1) {         L1: DADDIU    R3, R1, #-1
    ...              BNEZ      R3, L2      ; branch b2   (d != 1)
    L2

```

**Simple code  
fragment**

**Corresponding MIPS code:  
d is in R1**

Initial Value of d	Values of d				
	d==0?	b1	before b2	d==1?	b2
0	yes	not taken	1	yes	not taken
1	no	taken	1	yes	not taken
2	no	taken	2	no	taken

**Possible execution sequences assuming d is one of 0, 1, or 2**

# Correlating Predictors: Impact of Ignoring Correlation

Initial Value of d	d==0?	b1	Values of d before b2	d==1?	b2
0	yes	not taken	1	yes	not taken
1	no	taken	1	yes	not taken
2	no	taken	2	no	taken

**Possible execution sequences assuming d is one of 0, 1, or 2**

d=	b1 prediction	b1 action	new b1 prediction	b2 prediction	b2 action	new b2 prediction
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

**Behavior of 1-bit predictor initialized to not taken with d alternating between 2 and 0: 100% misprediction!**

T1: Branch History Table  
if last branch taken

1
0
....

T2: Branch History Table  
if last branch not taken

1
1
....

Branch Address



Last Branch Result



Branch Prediction

# Correlating Predictors: Taking Correlation into Account

Prediction bits	Prediction if last branch not taken	Prediction if last branch taken
NT/NT	NT	NT
NT/T	NT	T
T/NT	T	NT
T/T	T	T

**Meaning of 1-bit predictor with 1 bit of correlation: equivalent to assuming two separate prediction bits – one assuming last branch executed was not taken and one assuming the last branch executed was taken**

d=	b1 prediction	b1 action	new b1 prediction	b2 prediction	b2 action	new b2 prediction
2	NT/NT	T	T/NT	NT/NT	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	NT/T
2	T/NT	T	T/NT	NT/T	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	NT/T

**Behavior of 1-bit predictor with 1-bit of correlation, assuming initially NT/NT and d alternating between 0 and 2: mispredictions only on first iteration.**

**Predictions used in red.**

# Correlating Predictors:

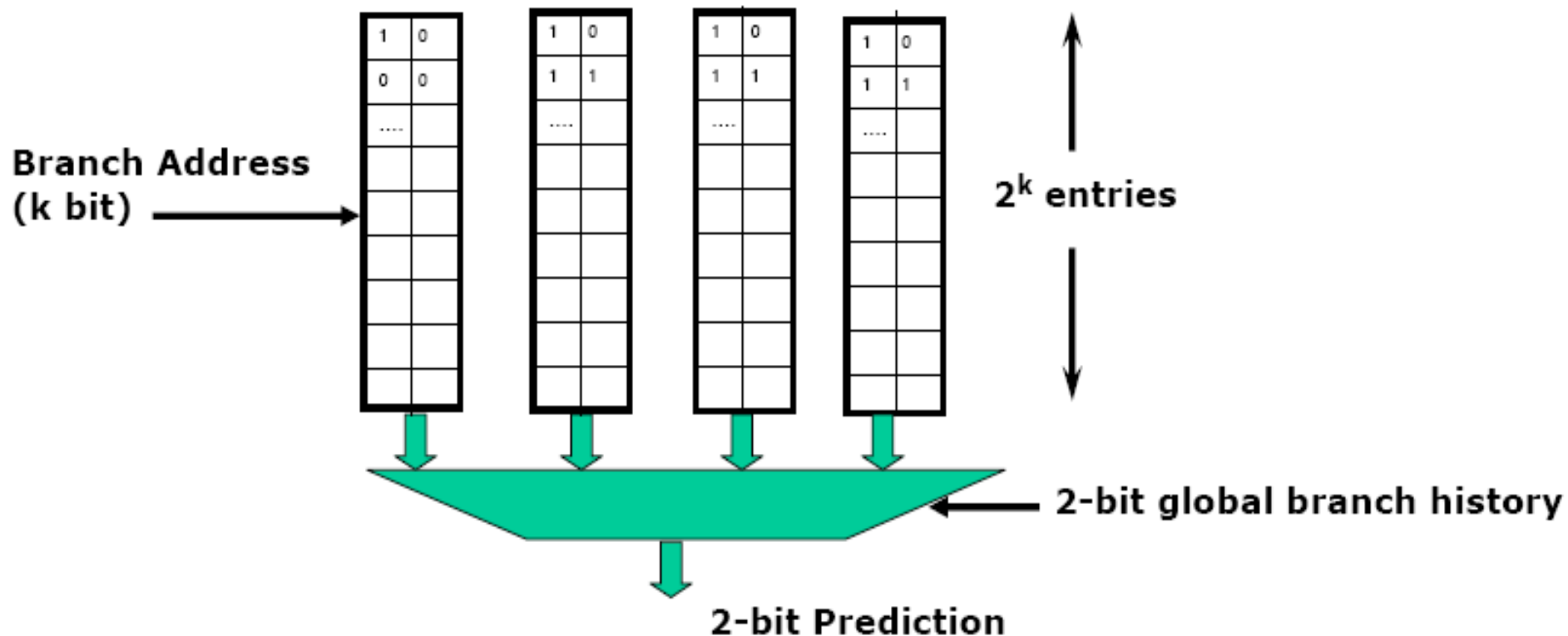
## ( $m,n$ ) Predictors

- The correlating predictor as before – 1 bit of prediction plus 1 correlating bit – is called a *(1,1) predictor*
- Generalization of the (1,1) predictor is the *( $m,n$ ) predictor*
- *( $m,n$ ) predictor* : use the *behavior of the last  $m$  branches* to choose from one of  $2^m$  branch predictors, *each of which is an  $n$ -bit predictor*
- The history of the most recent  $m$  branches is recorded in an  $m$ -bit shift register called the  $m$ -bit *global history register*
  - shift in the behavior bit for the most recent branch, shift out the the bit for the least recent branch
- Index into the BHT by concatenating the lower bits of the branch instruction address with the  $m$ -bit global history to access an  $n$ -bit entry



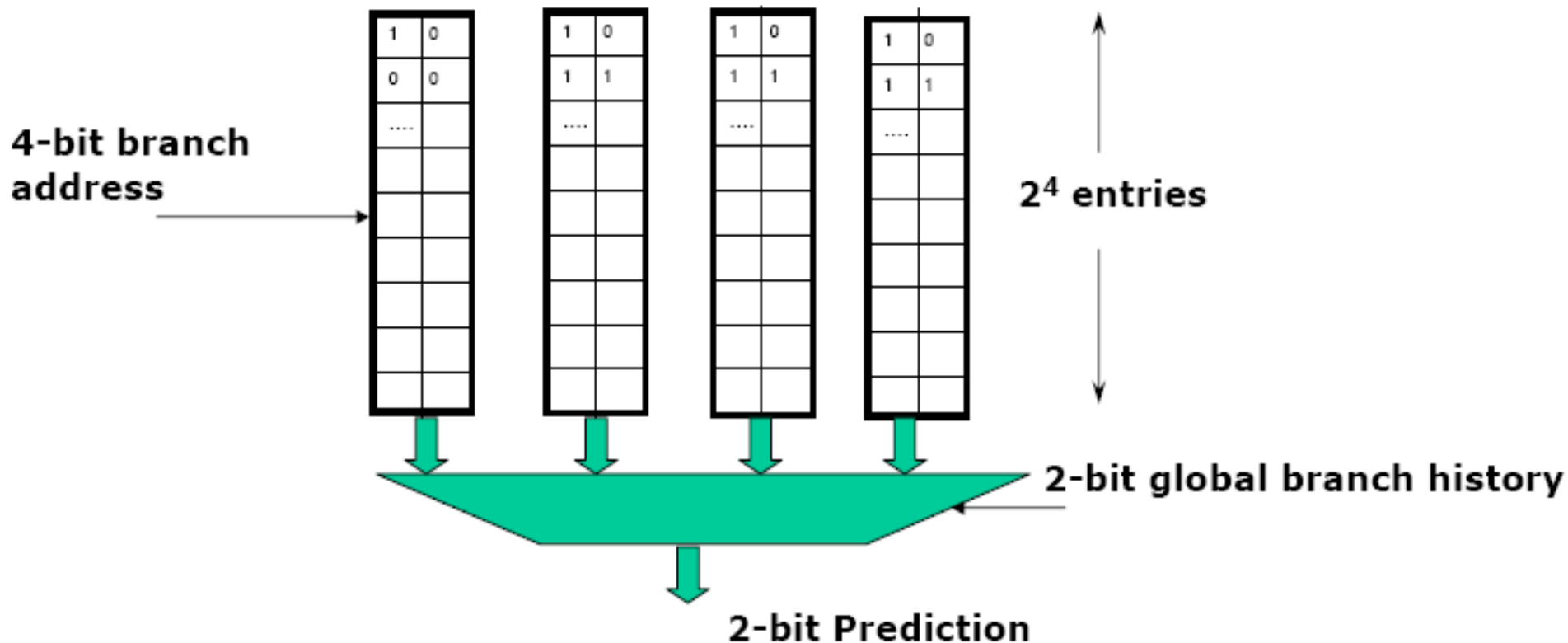
# (2, 2) Correlating Branch Predictors

- A (2, 2) correlating predictor has 4 2-bit Branch History Tables.
  - It uses the 2-bit global history to choose among the 4 BHTs.



# Example of (2, 2) Correlating Predictor

- Example: a (2, 2) correlating predictor with 64 total entries  $\Rightarrow$  6-bit index composed of: 2-bit global history and 4-bit low-order branch address bits



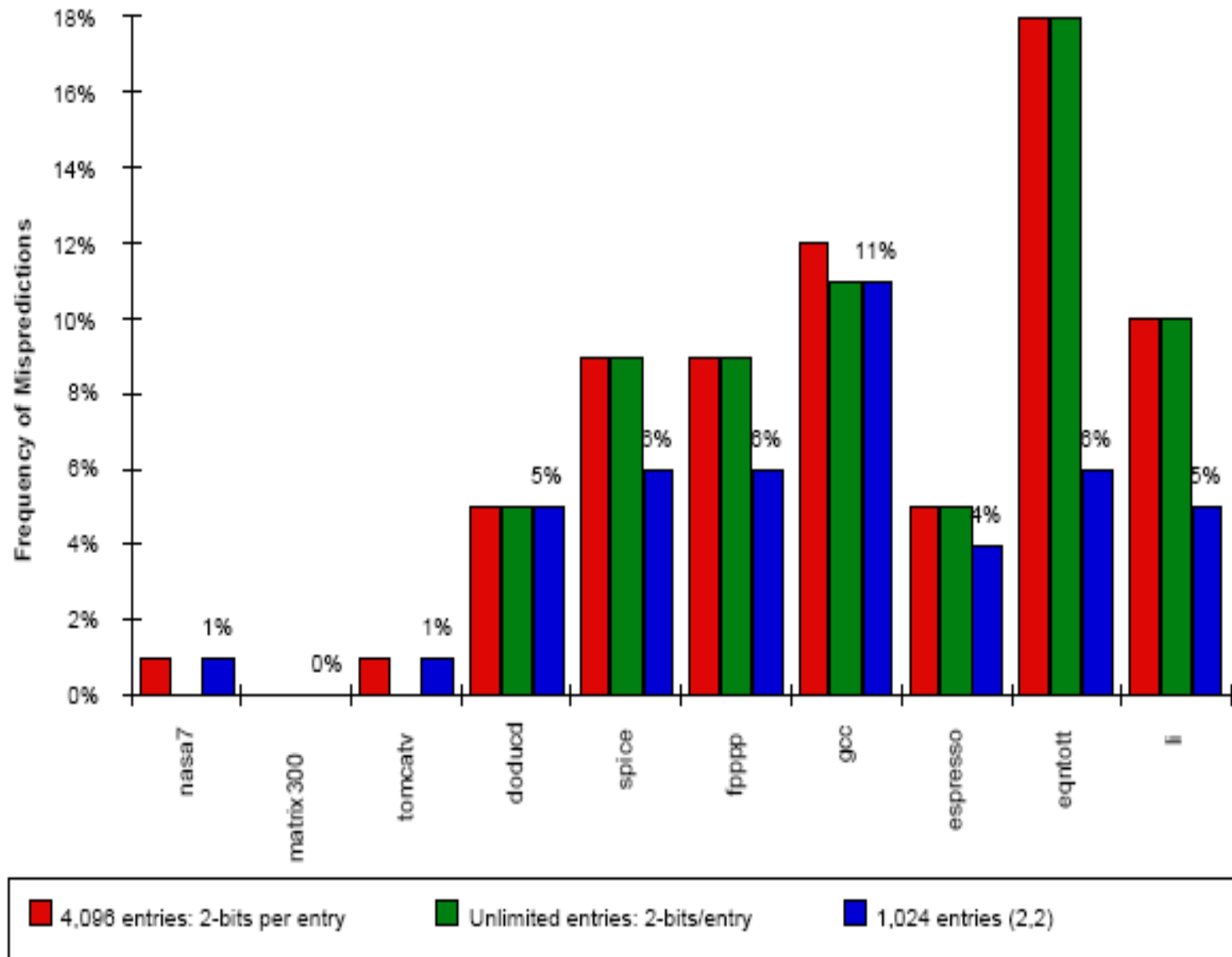
# Example of (2, 2) Correlating Predictor

- Each BHT is composed of 16 entries of 2-bit each.
- The 4-bit branch address is used to choose four entries (a row).
- 2-bit global history is used to choose one of four entries in a row (one of four BHTs)

# Accuracy of Correlating Predictors

- A 2-bit predictor with no global history is simply a (0, 2) predictor.
- By comparing the performance of a 2-bit simple predictor with 4K entries and a (2,2) correlating predictor with 1K entries.
- The (2,2) predictor not only outperforms the simply 2-bit predictor with the same number of total bits (4K total bits), it often outperforms a 2-bit predictor with an unlimited number of entries.

# Accuracy of Correlating Predictors



## 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

