Improved Branch Predictors

The branch predictors we have looked at so far make *local* predictions —predictions based on how *this* branch has been decided in the past.

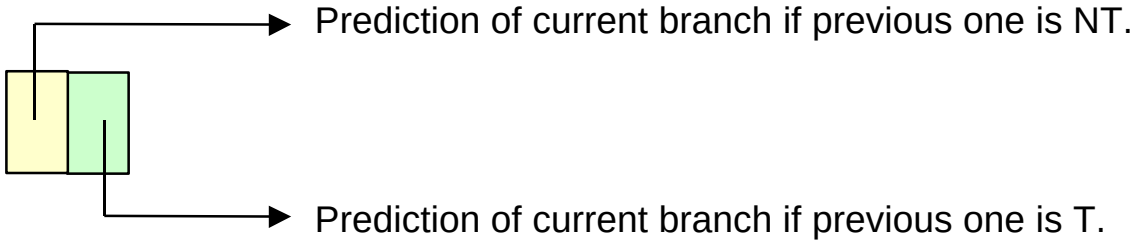
We might also look at *global* behavior, that is, how *all* recent branches have been decided.

H&P give an example of where this may be useful.

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

The outcome of B3 is correlated with the outcomes of B1 and B2. How?

We can design a branch predictor that predicts differently depending on the outcome of the previous branch.



In general, the last branch executed is *not* the same instruction as the branch being predicted.

(1, 1) correlating predictor

Let's consider a predictor that uses 1 bit of correlation. Each branch has two separate prediction bits, as shown above: One if the previous branch was T, and one if it was NT.

Consider this example.

```
B1: if (d == 0)
      d = 1;
B2: if (d == 1);
```

Assume that *d* alternates between 2 and 0.

d = ?	B1 prediction		B1 action	New B1 prediction		B2 prediction		B2 action	New B2 prediction	
2	**0**	0	1	1	0	0	**0**	1	0	1
0	1	**0**	0	1	0	**0**	1	0	0	1
2										
0										

How did this predictor do?

This predictor is called a (1, 1) predictor because it uses the behavior of the last branch to choose between a pair of 1-bit branch predictors.

(*m*, *n*) branch predictors

An (*m*, *n*) predictor uses the behavior of the last *m* branches to choose from among

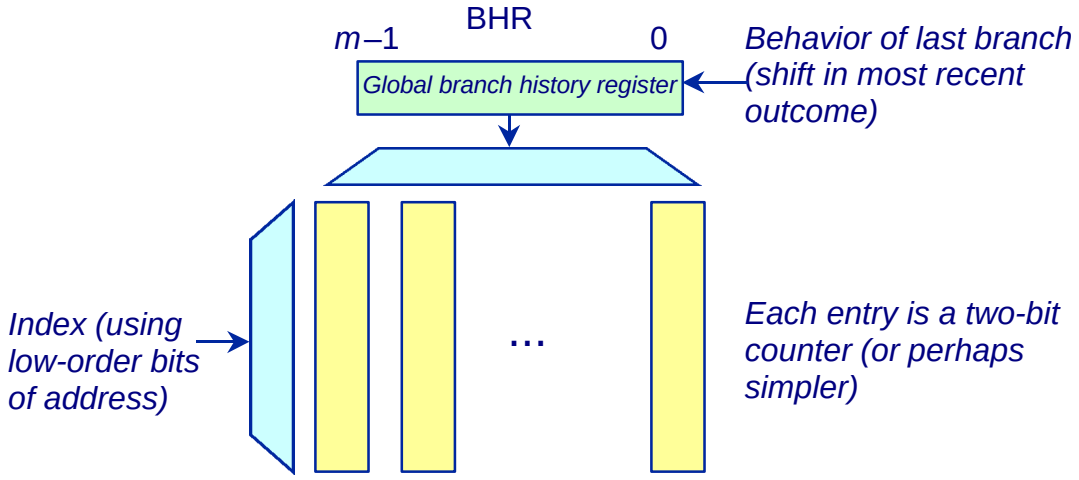
- 2^m branch predictors,
- each of which is an *n*-bit predictor for a single branch.

Gshare

The Gshare "global sharing" branch predictor is a (2, 2) predictor.

It uses both global branch history *and* the location of a branch instruction to create an index into a table of 2-bit counters.

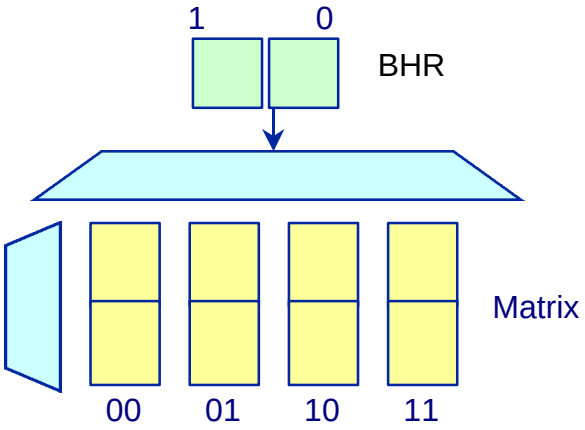
The 2-bit counters are like the ones at the end of the previous lecture.



Here is an example of how Gshare works.

Suppose that we have this code.

```
     Initially, r1 = 0
A:   BEQZ   R1, D
     …
D:   BEQZ   R1, F
     …
F:   NOT    R1, R1
G:   JUMP   A
```



All entries in the matrix are initialized to 01. In the tables below, matrix entries are italicized if they were updated due to the execution of the previous branch. The entry with the yellow background is the entry used for the prediction.

BHR = 00

	00	01	10	11
A:	01	01	01	01
D:	01	01	01	01

A: T, pred N
BHR → 01

BHR = 01

	00	01	10	11
A:	*10*	01	01	01
D:	01	01	01	01

D: T, pred N
BHR → 11

BHR = 11

	00	01	10	11
A:	10	01	01	01
D:	01	*10*	01	01

A: N, pred N
BHR → 10

BHR = 10

	00	01	10	11
A:	10	01	01	
D:	01	10	01	01

D: N, pred N
BHR → 00

BHR = 00

	00	01	10	11
A:	10	01	01	00
D:	01	10		01

A: T, pred T
BHR → 01

BHR = 01

	00	01	10	11
A:				
D:	01	10	00	01

D:
BHR →

BHR = 11

	00	01	10	11
A:	10	01	01	00
D:				

A: N, pred N
BHR → 10

BHR = 10

	00	01	10	11
A:	*11*	01	01	*00*
D:	01	10	00	01

D:
BHR →

The Gshare architecture uses an *m*-bit global history register to keep track of the direction of the last *m* executed branches.

To simplify the implementation, this global history register is **xor**ed with the (**PC>>2**) to create an index into a 2^m-entry pattern history table of *n*-bit counters.

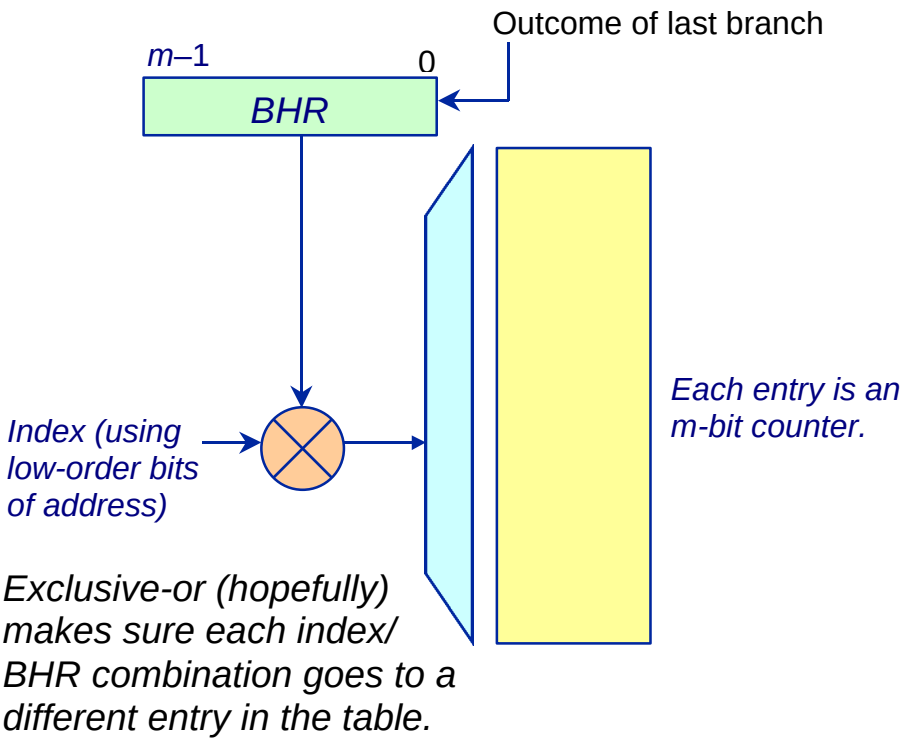
The result of this index is the prediction for the current branch. The predictor then compares this prediction with the real branch direction to determine if the branch was correctly predicted or not, and updates the prediction statistics.

The predictor then updates the *n*-bit counter used to perform the prediction. The counter is

- incremented if the branch was taken, and
- decremented if the branch was not taken.

Finally, the branch outcome is shifted into the most significant bit of the global history register:

```
global_reg = ((global_reg << 1) | direction)
```



Another way of implementing an (*m*, *n*) branch predictor is as a linear array that is *m* bits wide.

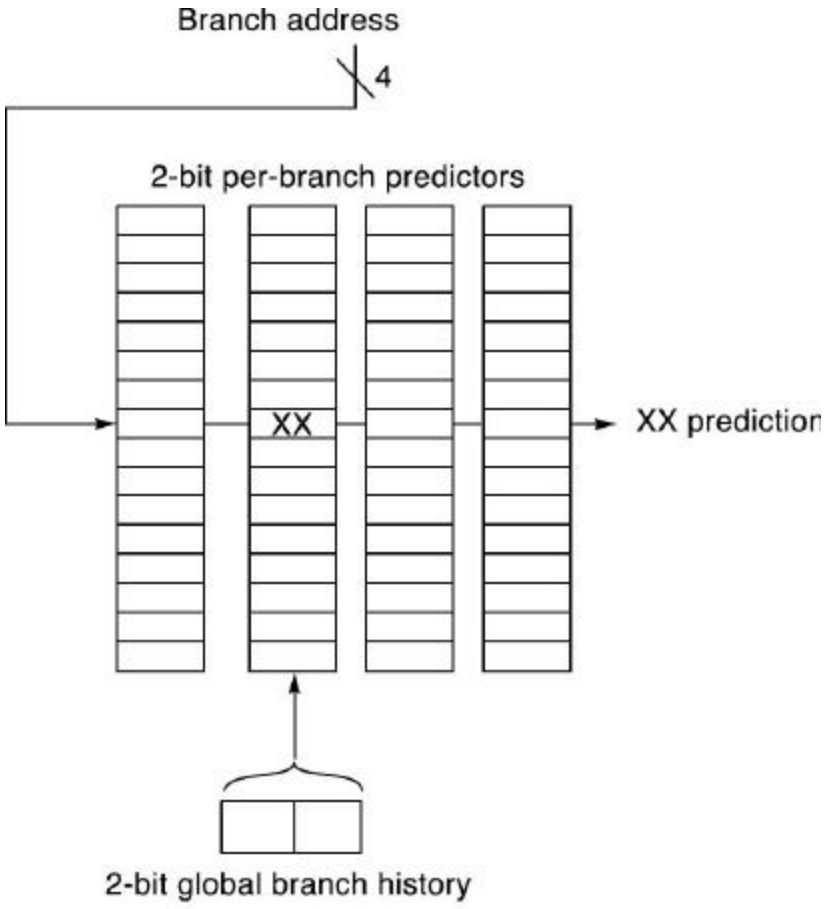
Indexing is done by concatenating the global history bits and the number of required bits from the branch address.

For example, the predictor shown below uses a 2-bit global history to choose from among four predictors for each branch address.

Each of these predictors is a 2-bit predictor for that branch.

This predictor has a total of 64 entries. An entry is selected by …

- the 4 low-order bits of the branch address (word address), and
- the two global history bits.



See Fig. 3.19 of H&P for evidence that a (2, 2) predictor with 1K entries far outperforms the (0, 2) predictor from the last lecture with 4K entries.

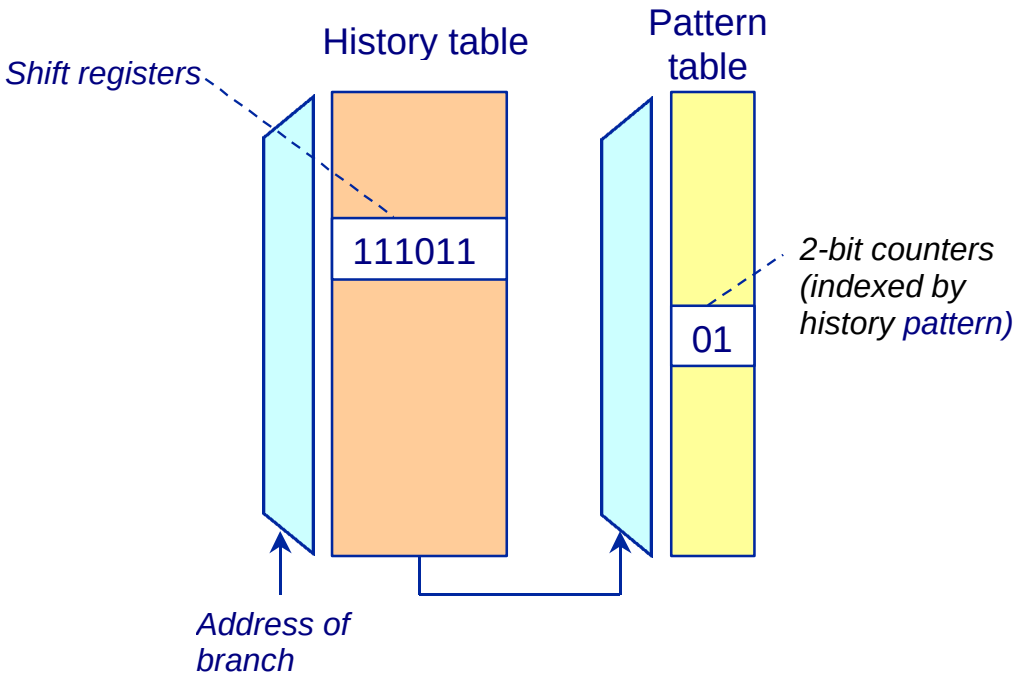
The Yeh-Patt predictor

The correlating branch predictors we have just studied work by combining local with global information.

However, it is also possible to do quite well considering only information about the current branch (*local* information).

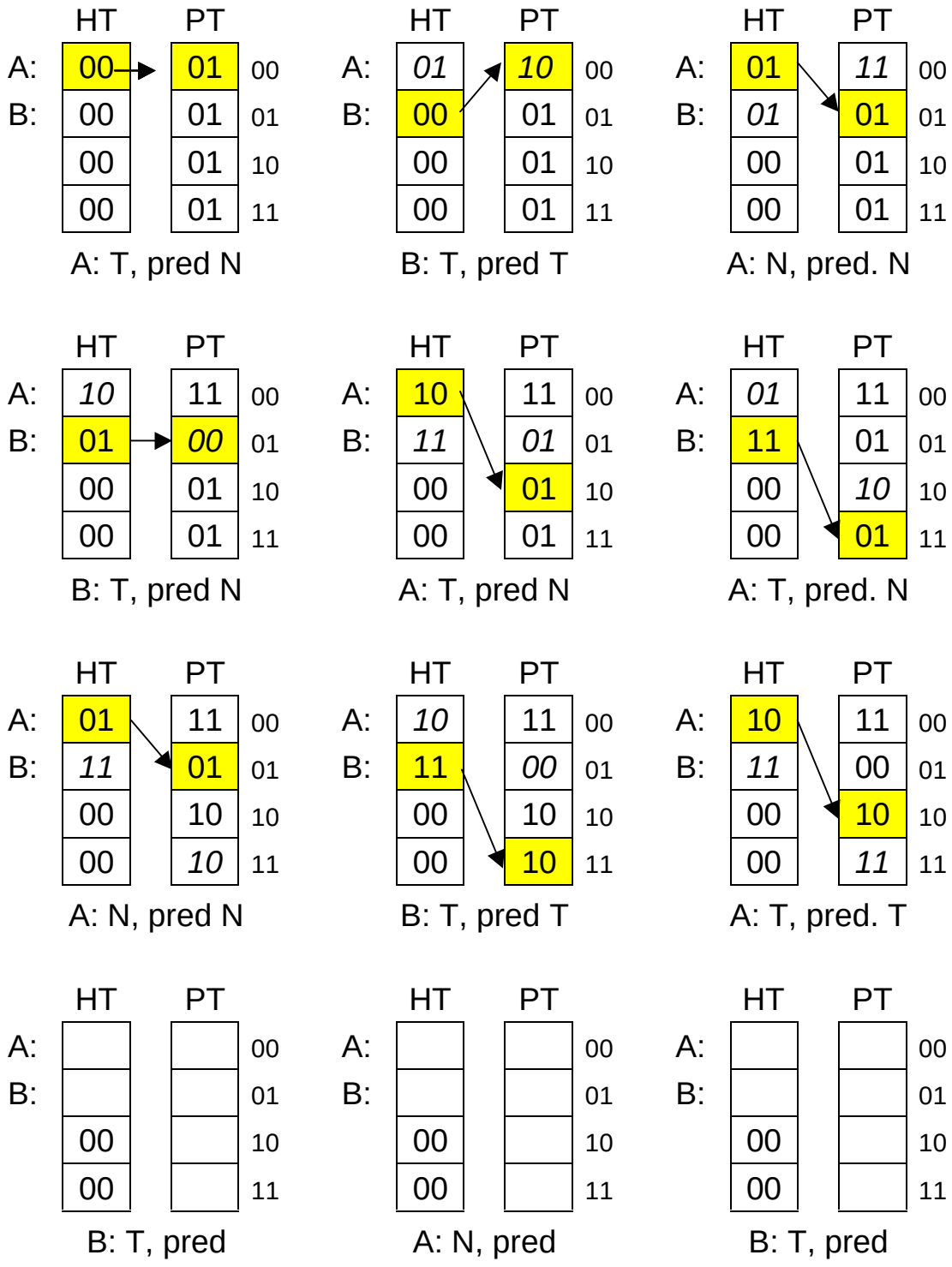
We do this by considering, What happened the last time a branch had the same history that the current branch does now?

This diagram shows how it operates.



Let us work out an example, assuming that two branches have this history:

A: T N T N T N T N
B: T T T T T T T T



PT entries 01 and 10 are “trained” for A, and ___ is “trained” for B.

In general, the Yeh-Patt predictor provides 96%–98% accuracy for integer code

The Alpha 21264: Combining local & global predictors

The correlating predictors we have studied combine the *global* branch history with *local* counters.

It is also possible to take local and global information into account like this:

- Create a branch predictor that uses only *local* information about a branch to predict the behavior of that branch.
- Create a predictor that uses only *global* information about all recent branches to predict the behavior of the next branch.
- Whenever it is time to predict a branch, choose the predictor (local or global) that "is working better" for that branch.

This strategy is employed in the Alpha 21264.

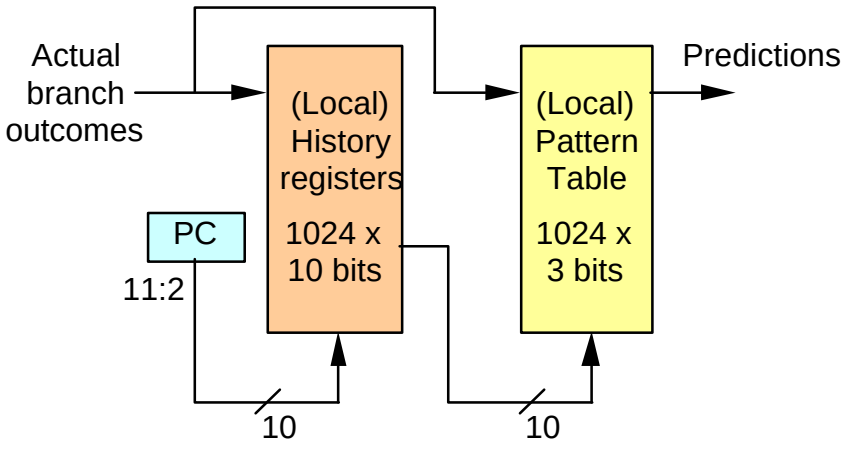
Because the hardware chooses between the better of two predictors at each point, it is called a *tournament predictor*.

Let's take a look at each of these predictors in more detail.

The local predictor

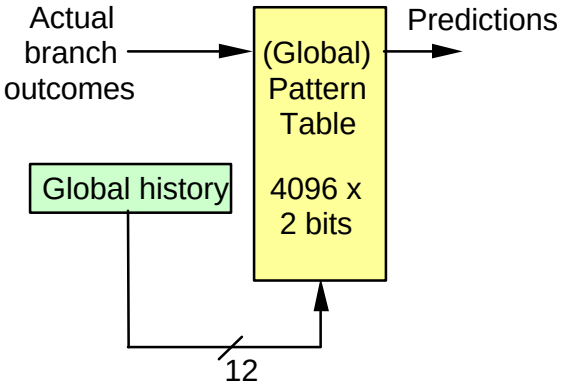
The local predictor is a Yeh-Patt predictor that holds the history of individual branches.

- Bits 11:2 of the program counter are used as the index to a 1K-entry table whose entries are 10-bit values, each representing the history of a particular branch.
- This 10-bit value is used as the index to a 1K-entry table of 3-bit saturating counters.
- The value of this saturating counter determines the local prediction.



The global predictor

The global predictor is indexed by a global history of the last 12 branches.

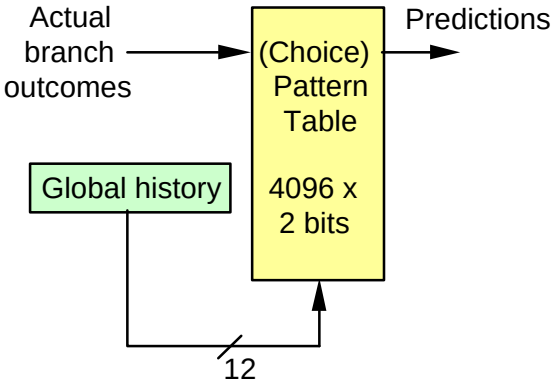


This 12-bit value is used as an index into a 4K-entry table of 2-bit saturating counters.

The value of the saturating counter determines the global prediction.

The choice predictor

The choice predictor monitors the history of the local and global predictors, and chooses the better of the two predictors for a particular branch.



The choice predictor is indexed by a global history of the last 12 branches.

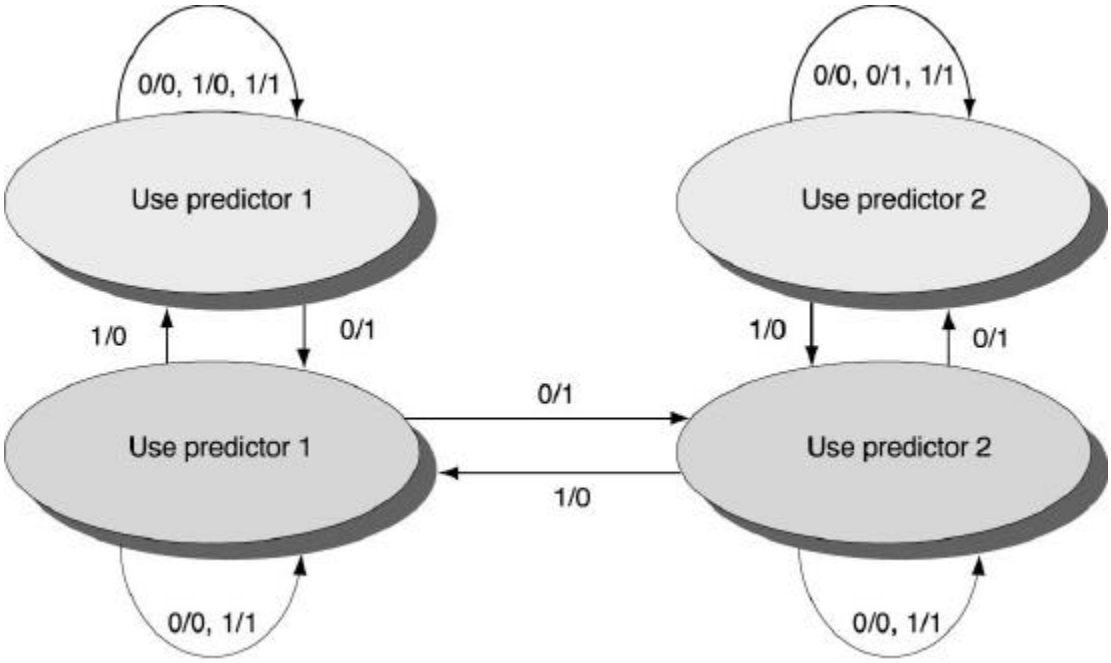
This 12-bit value is used as an index into a 4K-entry table of 2-bit saturating counters.

The value of the saturating counter determines whether to choose the local or global predictor.

How does the choice predictor determine this?

- Whenever the local counter is correct and the global counter is incorrect, the choice predictor's corresponding counter is decremented.
- Whenever the local counter is incorrect and the global counter is correct, the choice predictor's corresponding counter is incremented.
- If the counters are both correct, or both incorrect, the choice predictor's corresponding counter is not changed.

This leads to the following state-transition table:



Here is a diagram of how everything fits together.