

Dynamic Branch Prediction

Hung-Wei Tseng

Outline

- 2-level global predictor
- Hybrid predictors
- Perceptrons
- Branch and coding

Two-level global predictor

Marius Evers, Sanjay J. Patel, Robert S. Chappell, and Yale N. Patt. 1998. An analysis of correlation and predictability: what makes two-level branch predictors work. In Proceedings of the 25th annual international symposium on Computer architecture (ISCA '98).

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;  
do {  
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0  
        a[i] *= 2;  
    a[i] += i;  
} while ( ++i < 100) // Branch Y
```

(assume all states start with NT)

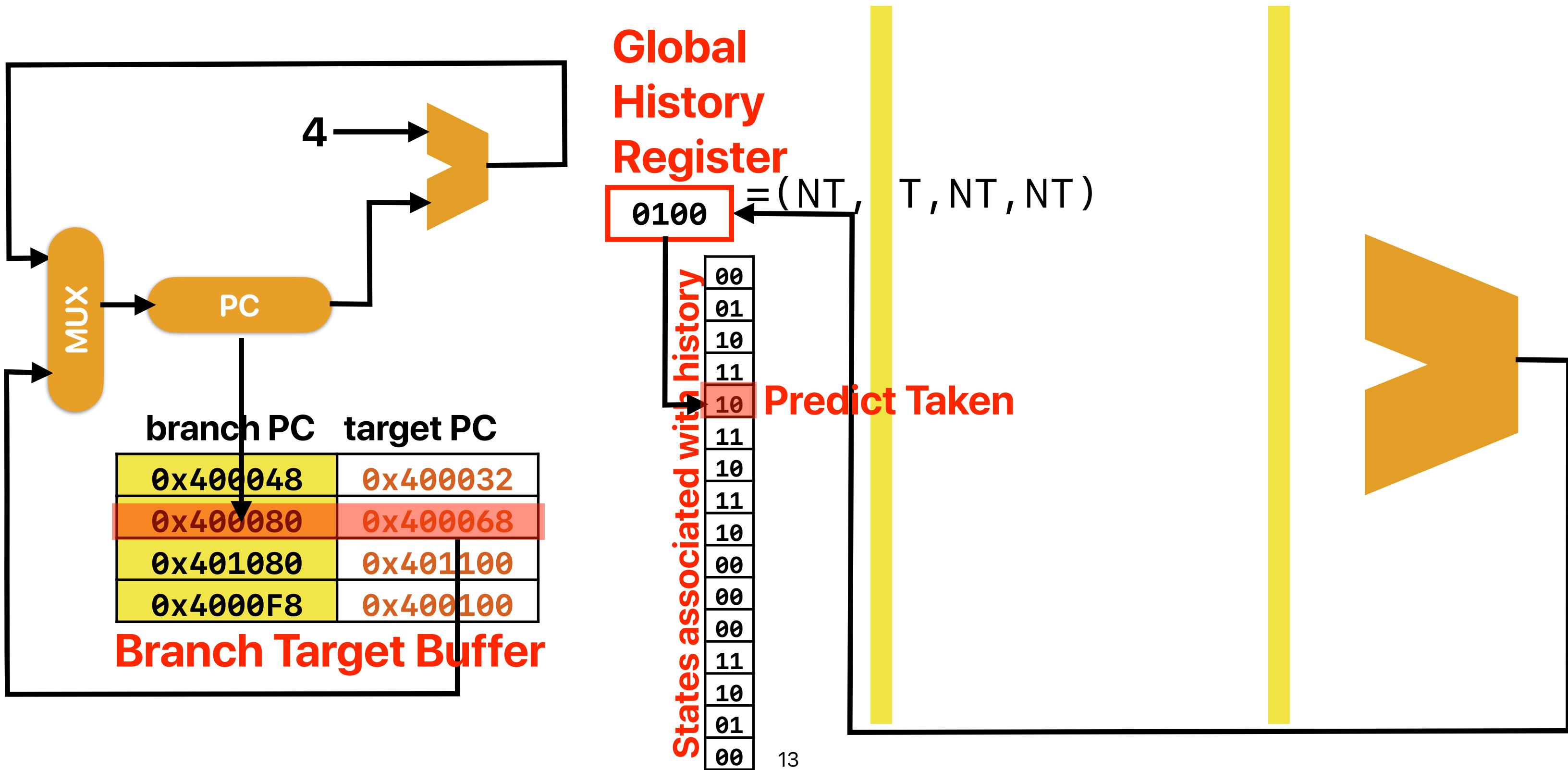
- A. ~25%
- B. ~33%
- C. ~50%
- D. ~67%
- E. ~75%

This pattern repeats all the time!

For branch Y, almost 100%,
For branch X, only 50%

i	branch?	state	prediction	actual
0	X	00	NT	T
1	Y	00	NT	T
2	X	01	NT	NT
2	Y	01	NT	T
3	X	00	NT	T
3	Y	10	T	T
3	X	01	NT	NT
4	Y	11	T	T
4	X	00	NT	T
5	Y	11	T	T
5	X	01	NT	NT
6	Y	11	T	T
6	X	00	NT	T
7	Y	11	T	T

Global history (GH) predictor



Performance of GH predictor

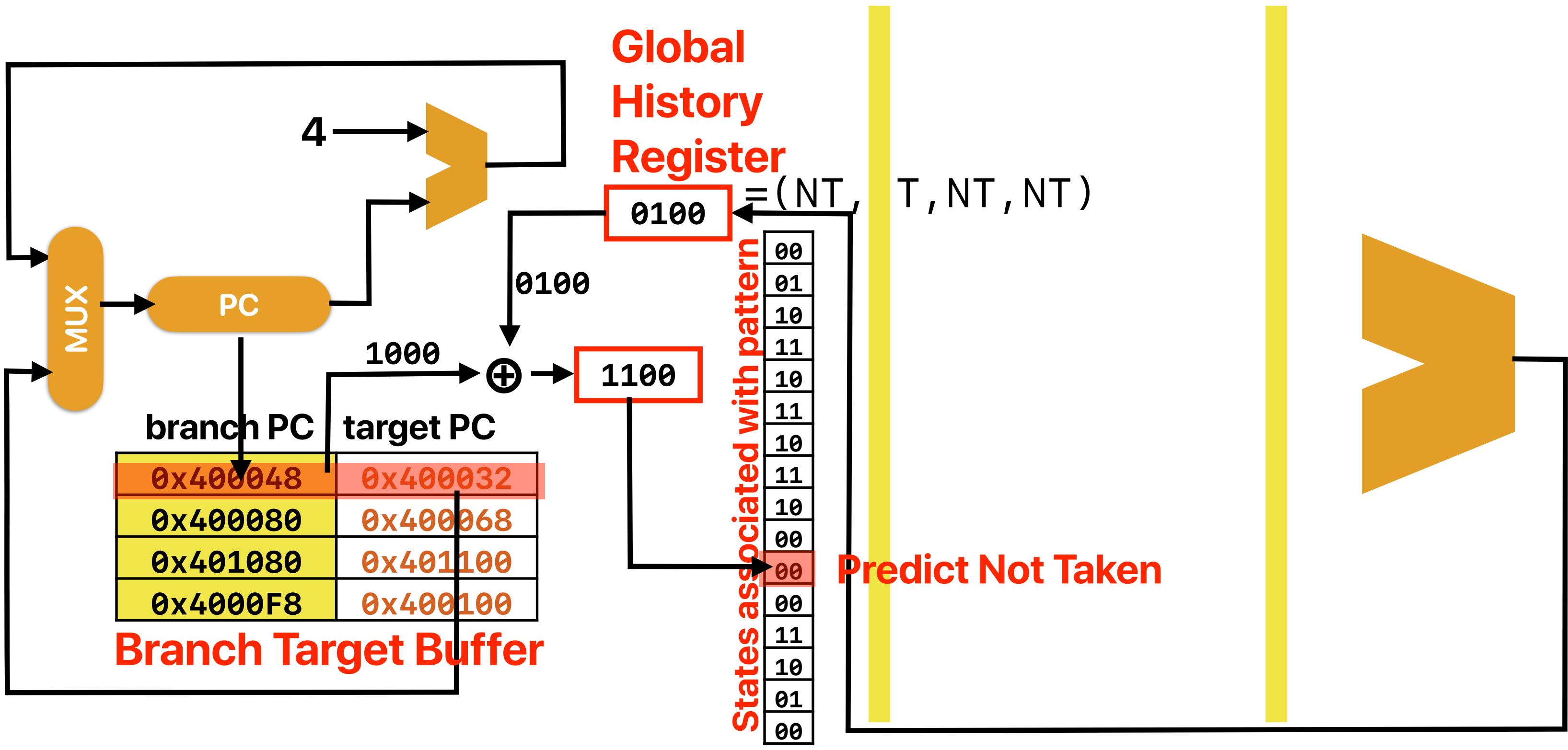
```
i = 0;  
do {  
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0  
        a[i] *= 2;  
    a[i] += i;  
} while ( ++i < 100)// Branch Y
```

Near perfect after this

i	branch?	GHR	state	prediction	actual
0	X	000	00	NT	T
1	Y	001	00	NT	T
1	X	011	00	NT	NT
2	Y	110	00	NT	T
2	X	101	00	NT	T
3	Y	011	00	NT	T
3	X	111	00	NT	NT
4	Y	110	01	NT	T
4	X	101	01	NT	T
5	Y	011	01	NT	T
5	X	111	00	NT	NT
6	Y	110	10	T	T
6	X	101	10	T	T
7	Y	011	10	T	T
7	X	111	00	NT	NT
8	Y	110	11	T	T
8	X	101	11	T	T
9	Y	011	11	T	T
9	X	111	00	NT	NT
10	Y	110	11	T	T
10	X	101	11	T	T
11	Y	011	11	T	T

Hybrid predictors

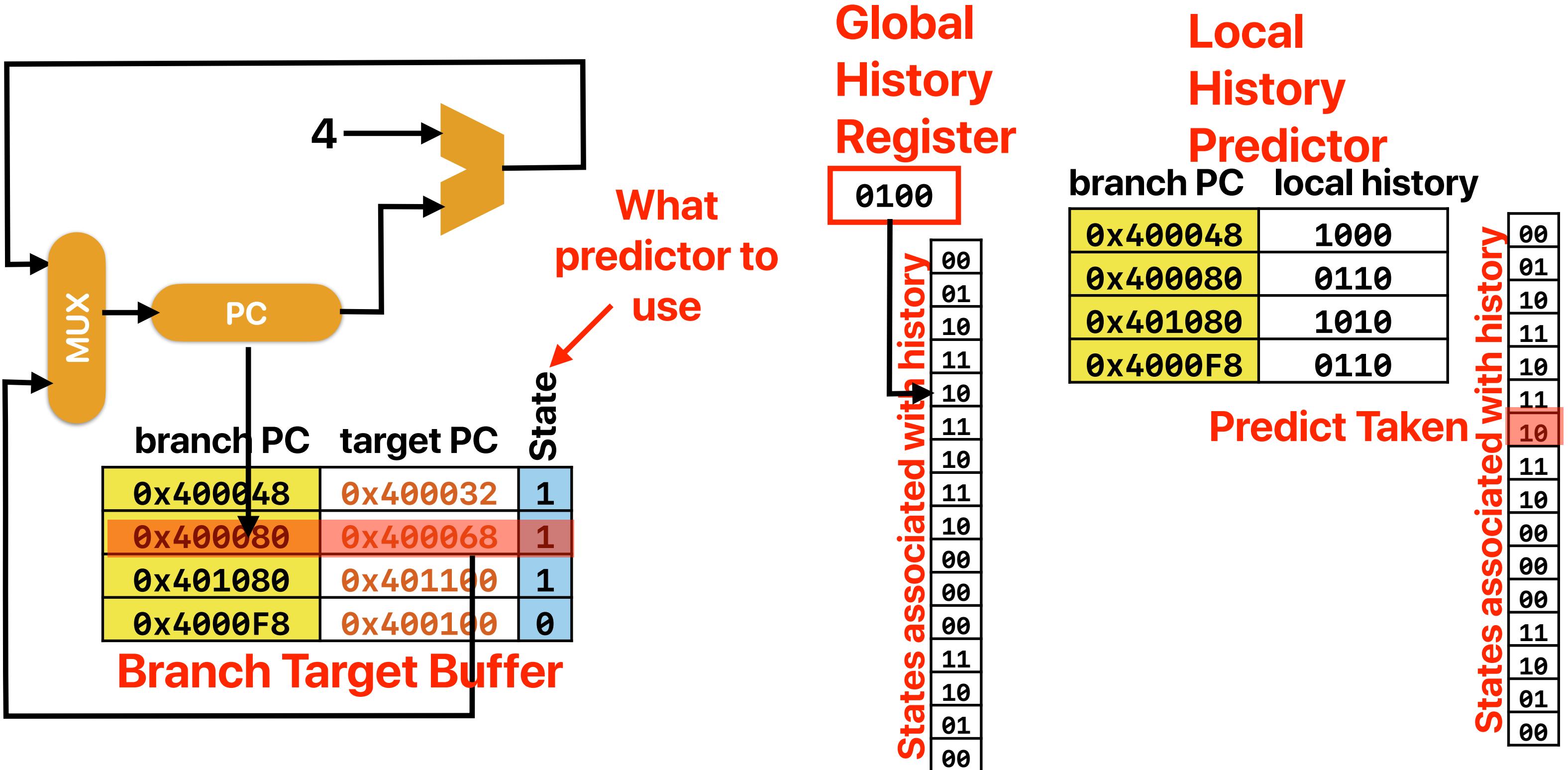
gshare predictor



gshare predictor

- Allowing the predictor to identify both branch address but also use global history for more accurate prediction

Tournament Predictor



Tournament Predictor

- The state predicts “which predictor is better”
 - Local history
 - Global history
- The predicted predictor makes the prediction

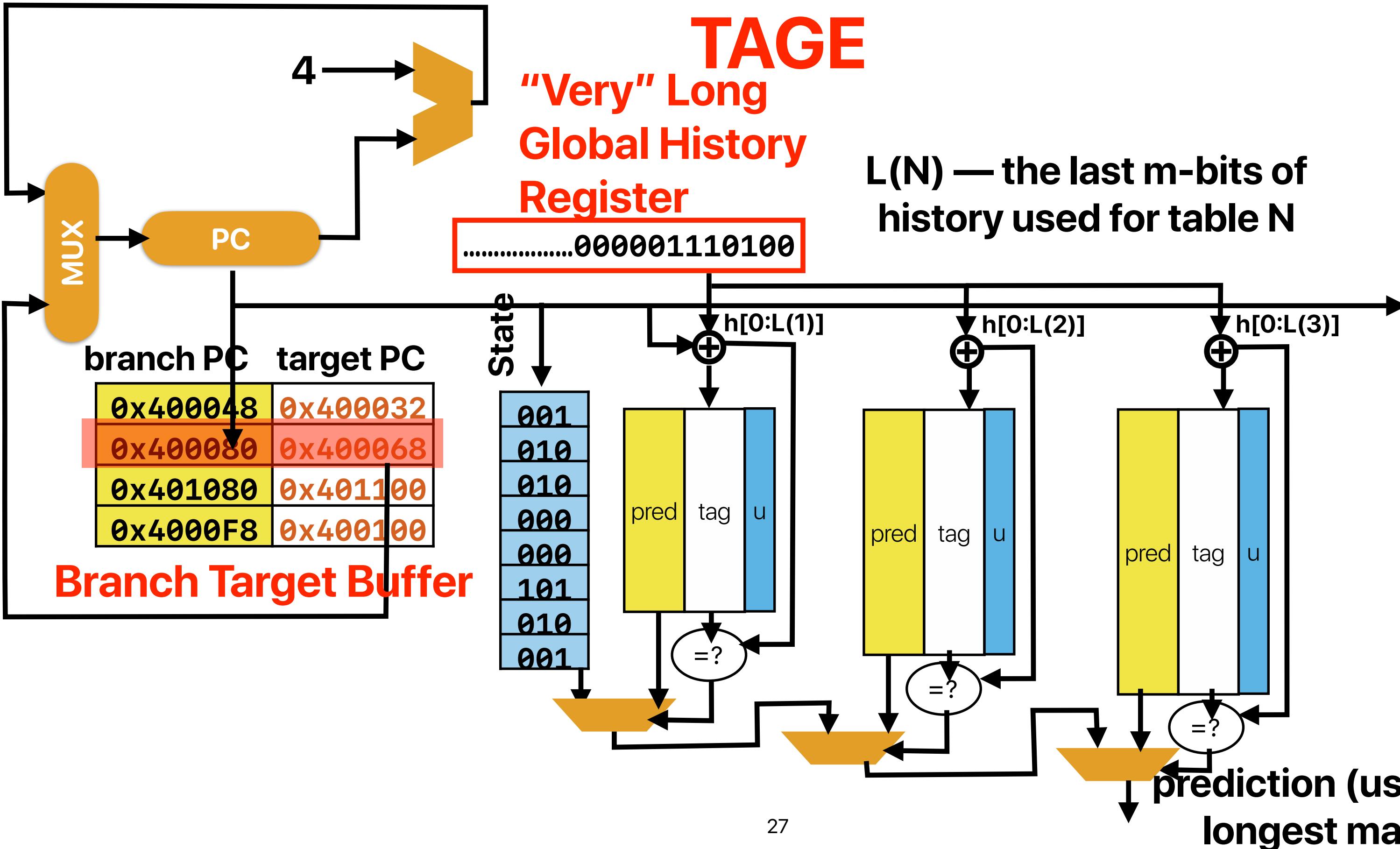
TAGE

André Seznec. The L-TAGE branch predictor. Journal of Instruction Level Parallelism (<http://wwwjilp.org/vol9>), May 2007.

TAGE

"Very" Long Global History Register

$L(N)$ — the last m -bits of history used for table N



Perceptron

Jiménez, Daniel, and Calvin Lin. "Dynamic branch prediction with perceptrons." Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture. IEEE, 2001.

The following slides are excerpted from <https://www.jilp.org/cbp/Daniel-slides.PDF> by Daniel Jiménez

Branch Prediction is Essentially an ML Problem

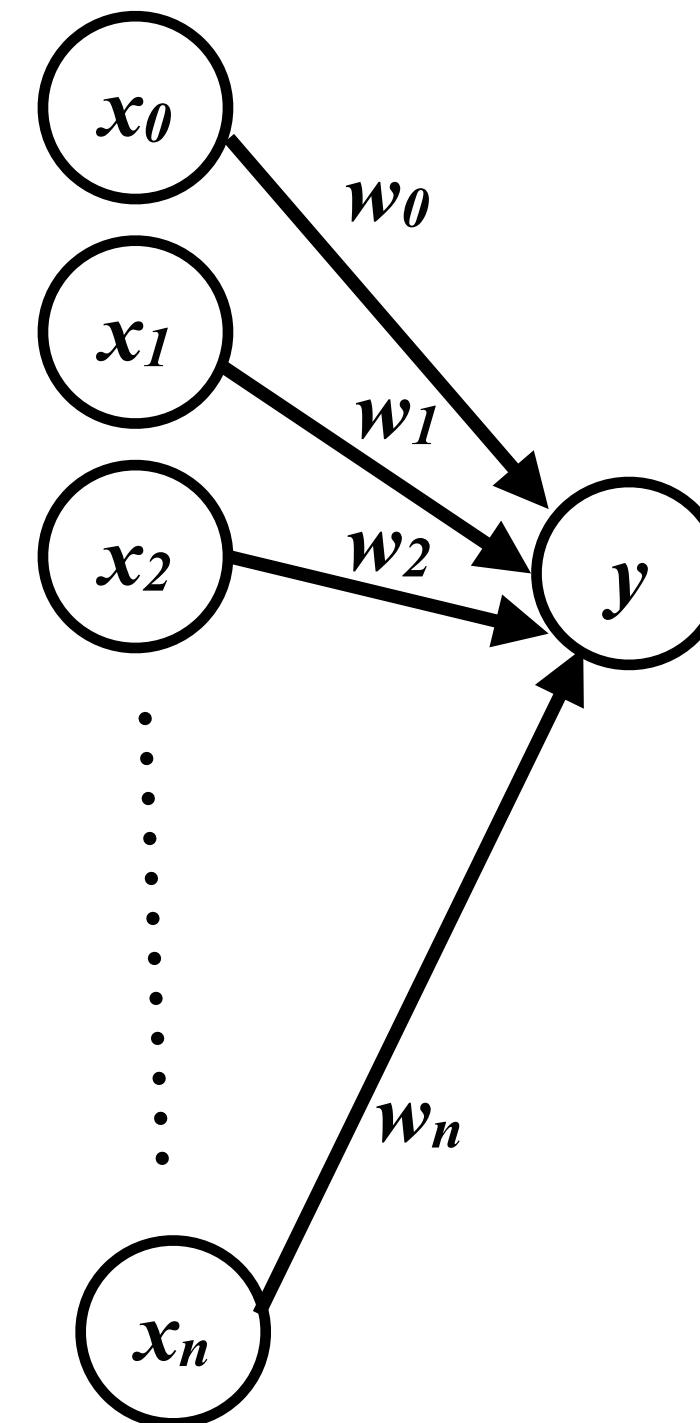
- The machine learns to predict conditional branches
- Artificial neural networks
 - Simple model of neural networks in brain cells
 - Learn to recognize and classify patterns

Mapping Branch Prediction to NN

- The inputs to the perceptron are branch outcome histories
 - Just like in 2-level adaptive branch prediction
 - Can be global or local (per-branch) or both (alloyed)
 - Conceptually, branch outcomes are represented as
 - +1, for taken
 - -1, for not taken
- The output of the perceptron is
 - Non-negative, if the branch is predicted taken
 - Negative, if the branch is predicted not taken
 - Ideally, each static branch is allocated its own perceptron

Mapping Branch Prediction to NN (cont.)

- Inputs (x 's) are from branch history and are -1 or +1
- $n + 1$ small integer weights (w 's) learned by on-line training
- Output (y) is dot product of x 's and w 's; predict taken if $y \geq 0$
- Training finds correlations between history and outcome



$$y = w_0 + \sum_{i=1}^n x_i w_i$$

Training Algorithm

$x_{1..n}$ is the n -bit history register, x_0 is 1.

$w_{0..n}$ is the weights vector.

t is the Boolean branch outcome.

θ is the training threshold.

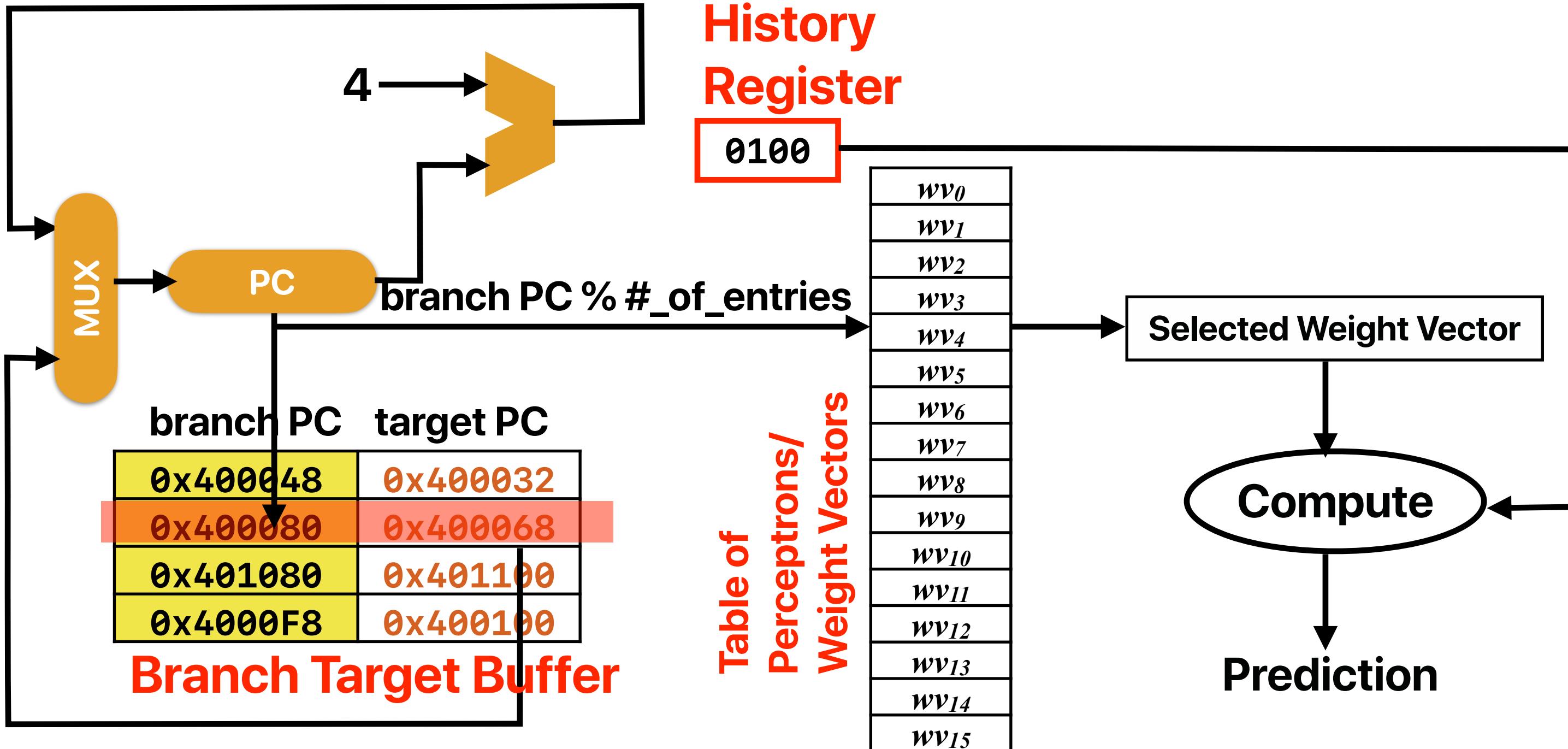
```
if  $|y| \leq \theta$  or  $((y \geq 0) \neq t)$  then
    for each  $0 \leq i \leq n$  in parallel
        if  $t = x_i$  then
             $w_i := w_i + 1$ 
        else
             $w_i := w_i - 1$ 
        end if
    end for
end if
```

Predictor Organization

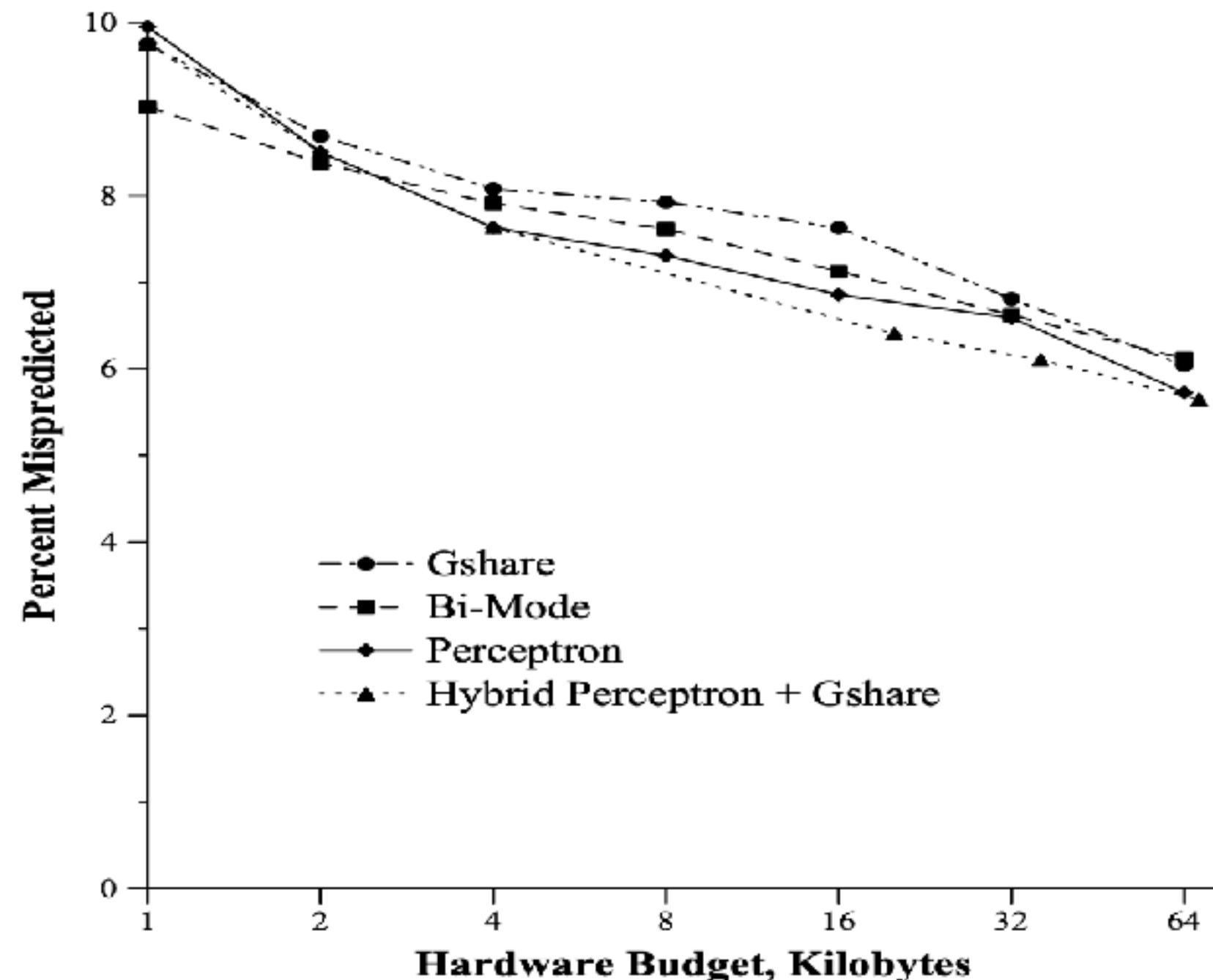
Global

History
Register

0100



How good is prediction using perceptrons?



Perceptron vs. other techniques, Context Switching

How good is prediction using perceptrons?

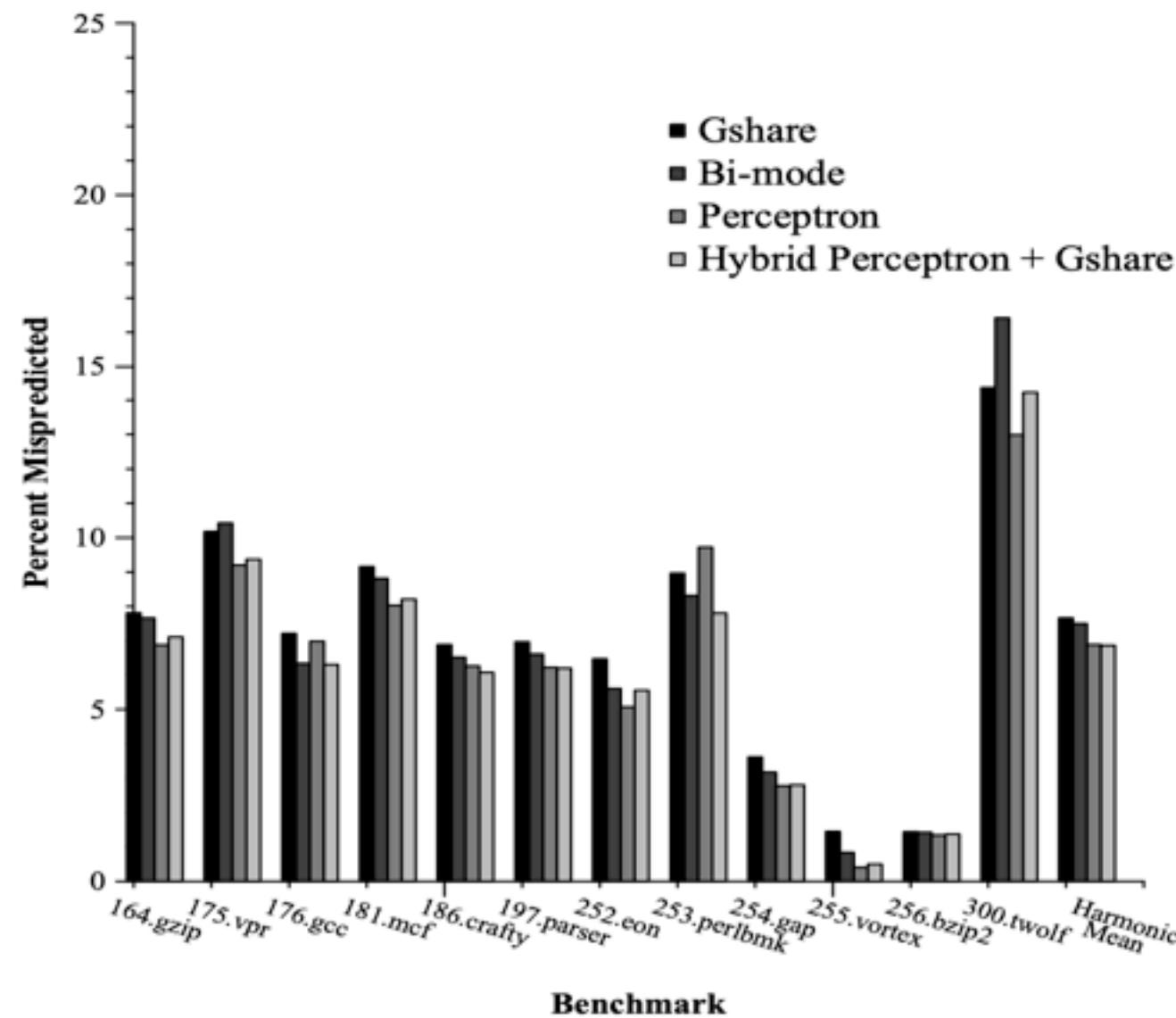


Figure 4: Misprediction Rates at a 4K budget. The perceptron predictor has a lower misprediction rate than *gshare* for all benchmarks except for *186.crafty* and *197.parser*.

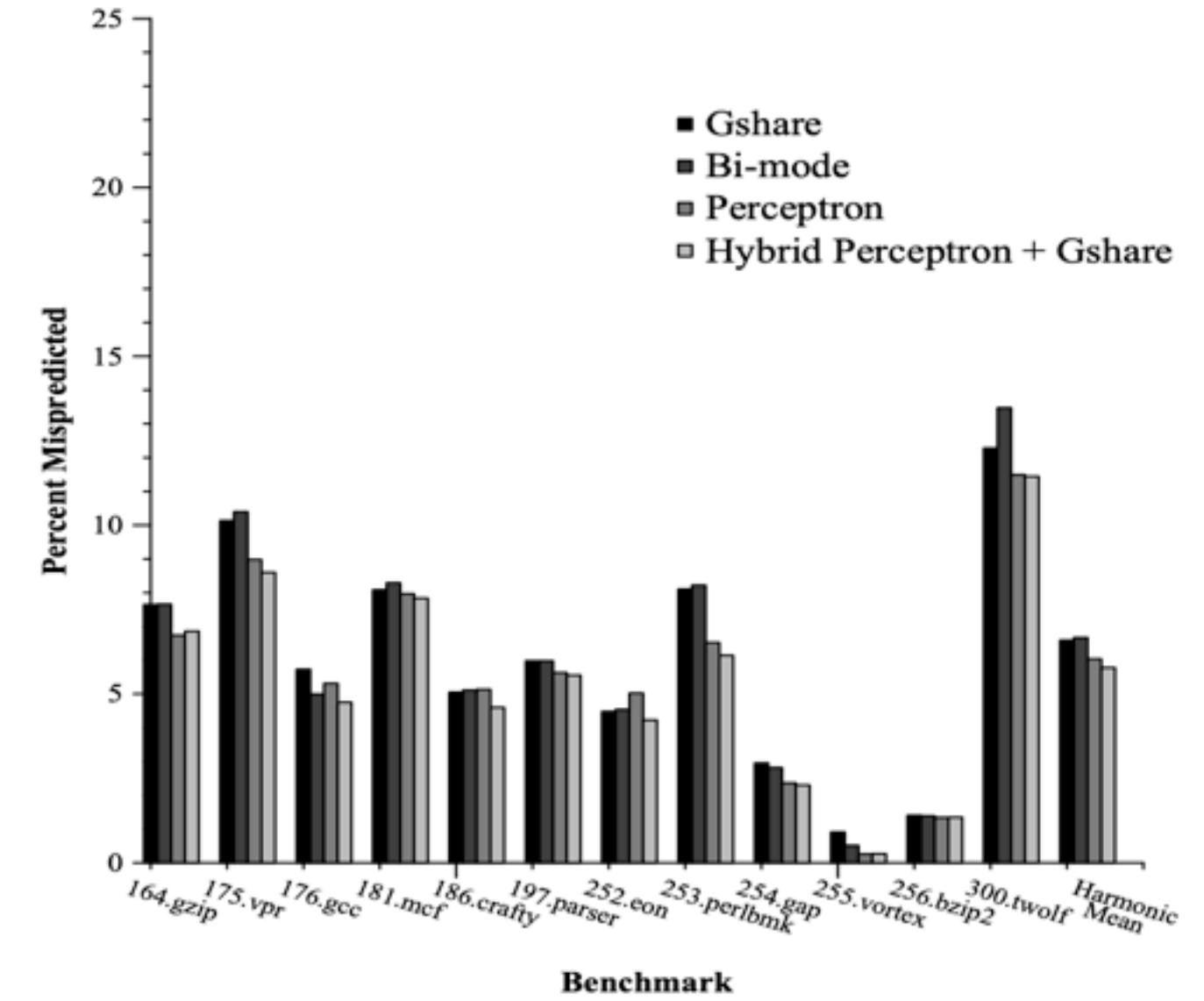
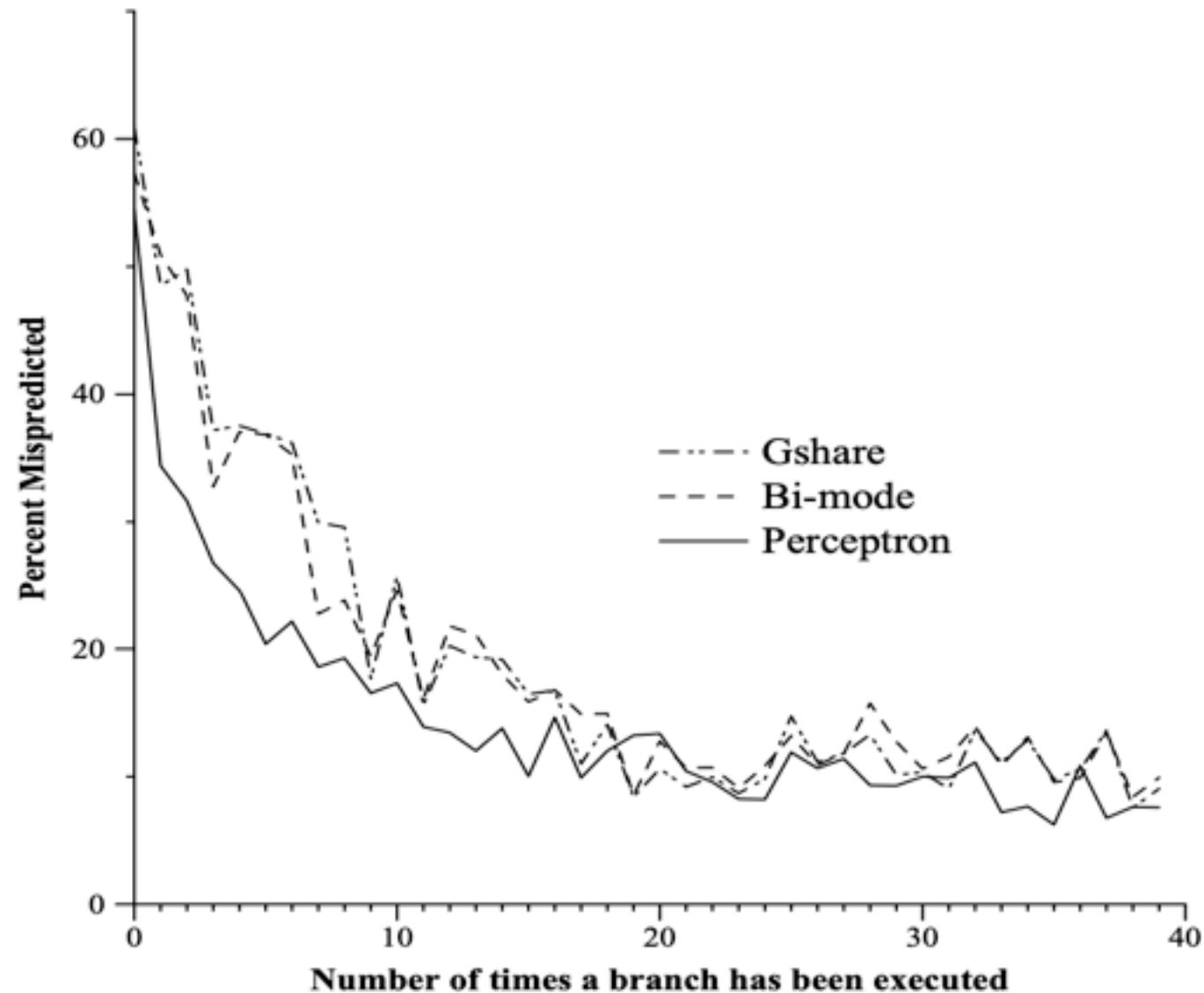


Figure 5: Misprediction Rates at a 16K budget. *Gshare* outperforms the perceptron predictor only on *186.crafty*. The hybrid predictor is consistently better than the PHT schemes.

History/training for perceptrons



Hardware budget in kilobytes	History Length		
	<i>gshare</i>	bi-mode	perceptron
1	6	7	12
2	8	9	22
4	8	11	28
8	11	13	34
16	14	14	36
32	15	15	59
64	15	16	59
128	16	17	62
256	17	17	62
512	18	19	62

Table 1: Best History Lengths. This table shows the best amount of global history to keep for each of the branch prediction schemes.

Branch predictors in processors

- The Intel Pentium MMX, Pentium II, and Pentium III have local branch predictors with a local 4-bit history and a local pattern history table with 16 entries for each conditional jump.
- Global branch prediction is used in Intel Pentium M, Core, Core 2, and Silvermont-based Atom processors.
- Tournament predictor is used in DEC Alpha, AMD Athlon processors
- The AMD Ryzen multi-core processor's Infinity Fabric and the Samsung Exynos processor include a perceptron based neural branch predictor.

Branch and programming

Demo revisited

- Why the sorting the array speed up the code despite the increased instruction count?

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)
            sum++;
    }
}
```

Demo: Popcount

- The population count (or popcorn) of a specific value is the number of set bits (i.e., bits in 1s) in that value.
- Applications
 - Parity bits in error correction/detection code
 - Cryptography
 - Sparse matrix
 - Molecular Fingerprinting
 - Implementation of some succinct data structures like bit vectors and wavelet trees.

Demo: pop count

- Given a 64-bit integer number, find the number of 1s in its binary representation.
- Example 1:

Input: 9487

Output: 7

Explanation: 9487's binary representation is
0b1001010000111

```
int main(int argc, char *argv[]) {  
    uint64_t key = 0xdeadbeef;  
  
    int count = 1000000000;  
    uint64_t sum = 0;  
  
    for (int i=0; i < count; i++)  
    {  
        sum += __builtin_popcount(key);  
    }  
    printf("Result: %lu\n", sum);  
    return sum;  
}
```

Hardware acceleration

- Because `popcount` is important, both intel and AMD added a `POPCNT` instruction in their processors with SSE4.2 and SSE4a
- In C/C++, you may use the intrinsic “`_mm_popcnt_u64`” to get # of “1”s in an unsigned 64-bit number
 - You need to compile the program with `-m64 -msse4.2` flags to enable these new features

```
#include <smmintrin.h>
inline int popcount(uint64_t x) {
    int c = _mm_popcnt_u64(x);
    return c;
}
```