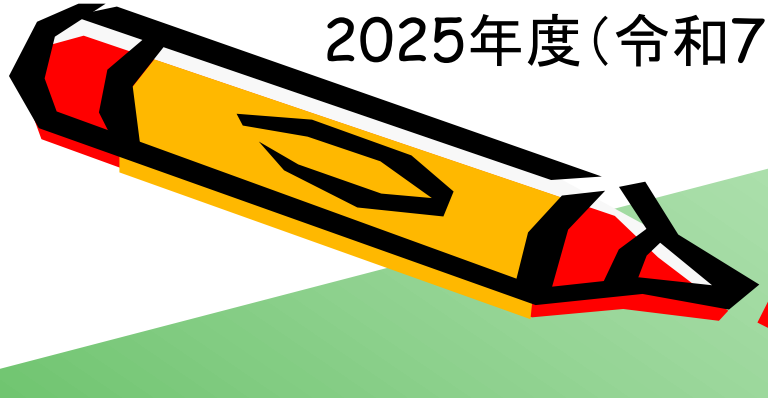


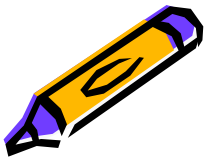
2025年度（令和7年）版

Ver. 2025-10-24a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

7. 分岐予測 (2)
Branch Prediction (2)



www.arch.cs.titech.ac.jp/lecture/CA/
Tue 13:30-15:10, 15:25-17:05
Fri 13:30-15:10

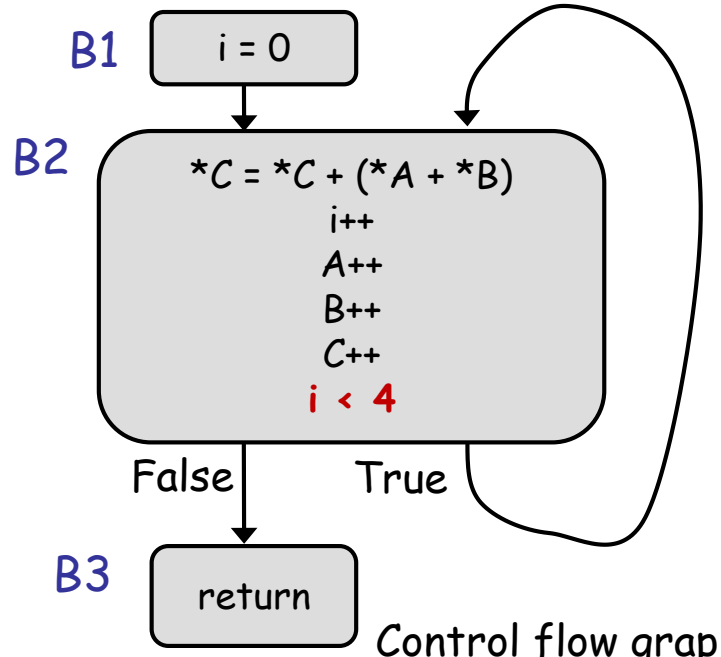
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

高いバンド幅の命令フェッチ

- パイプラインにバブルを生じさせないために，条件分岐命令をフェッチした時に次の3つを予測 (prediction) する.
  - フェッチしている命令が分岐命令か？
  - 分岐先アドレス(32bit)
  - 分岐方向
    - 分岐成立(Taken)か分岐不成立(Not taken, Untaken)か？
- Speculation assuming the prediction is true
  - No penalty by a branch instruction when the prediction hits
  - Recovery when the miss is detected

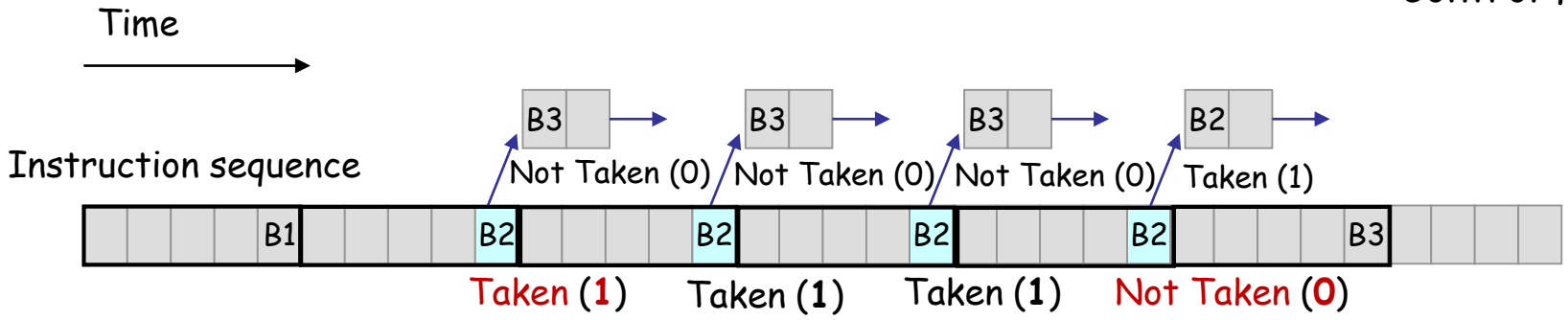
Sample program: vector add (function v_add)

```
#define VSIZE 4
void v_add(int *A, int *B, int *C){
  for(i=0; i<VSIZE; i++)
    C[i] += (A[i] + B[i]);
}
```



Control flow graph

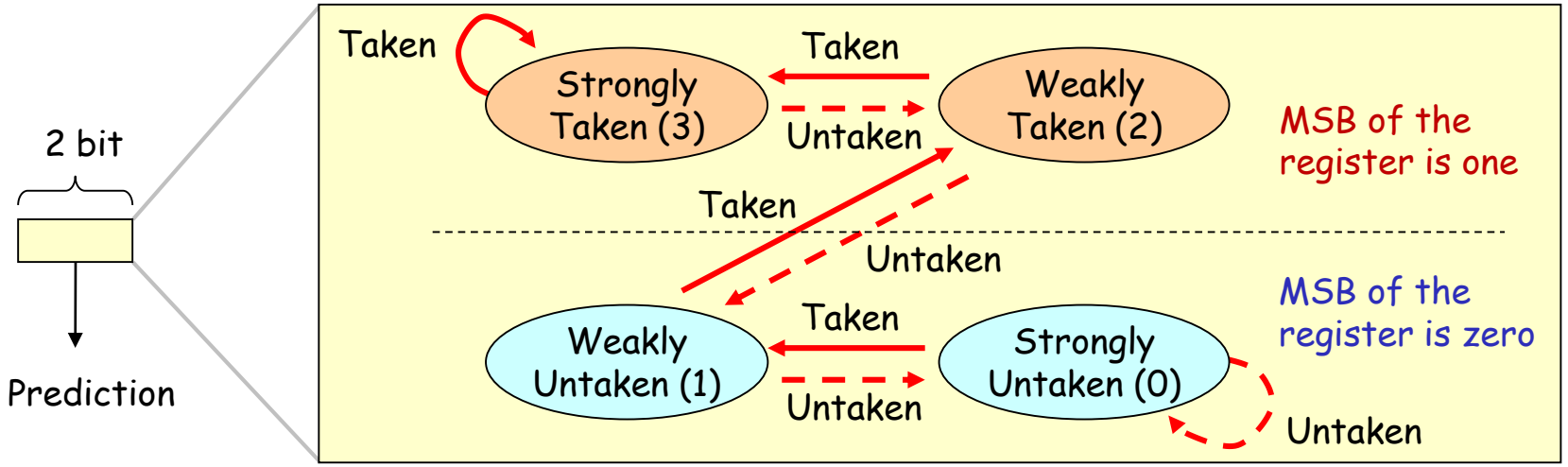
Basic block contains a sequence of statement. The flow of control enters at the beginning of the statement and leave at the end.



Predicting the branch outcome sequence of 1110 1110 1110 1110 1110 …

Simple branch predictor: 2-bit counter (2BC)

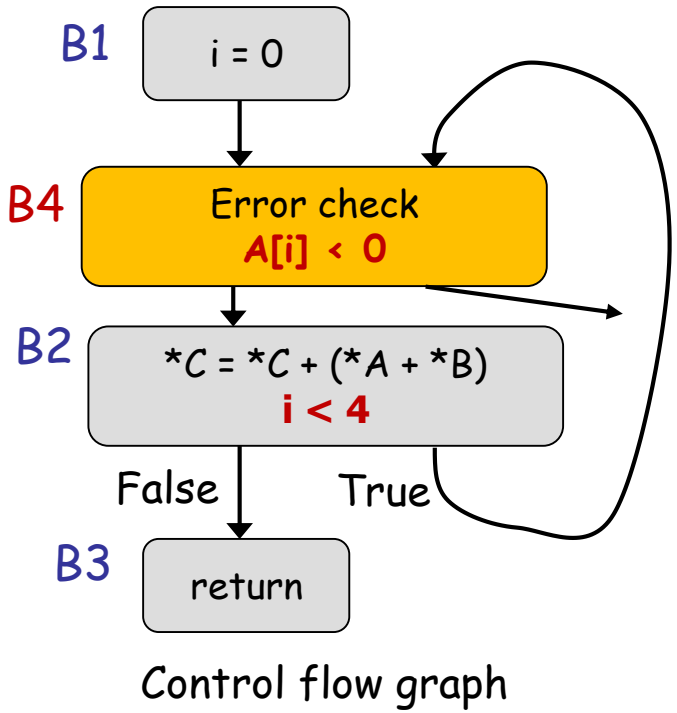
- It uses two bit register as a saturating counter.
- How to update the register
  - If the branch outcome is taken and the value is not 3, then increment the register.
  - If the branch outcome is untaken and the value is not 0, then decrement the register.
- Hot to predict
  - It predicts as 1 if the MSB of the register is one, otherwise predicts as 0.



Predicting the sequence of	1110 1110 1110 1110 1110 ...
State of the counter	2333 2333 2333 2333 2333 ...
Prediction	1111 1111 1111 1111 1111 ...
Hit/Miss of the pred.	HHHM HHHM HHHM HHHM HHHM

Sample program: vector add with two branches

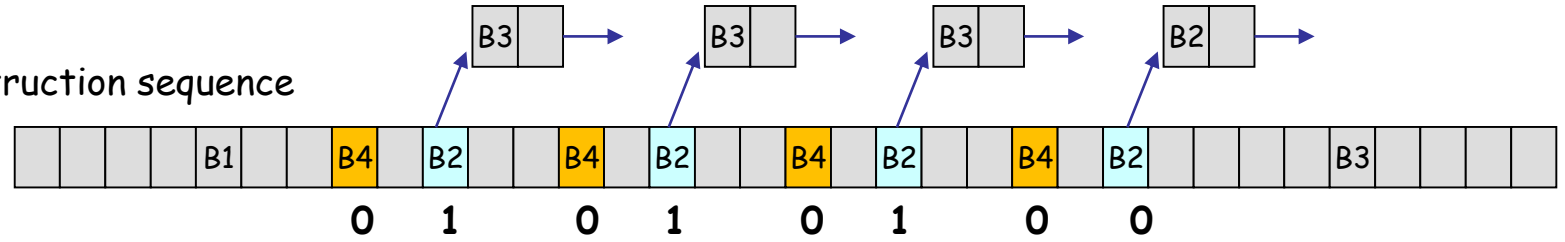
```
#define VSIZE 4
void v_add(int *A, int *B, int *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```



Control flow graph

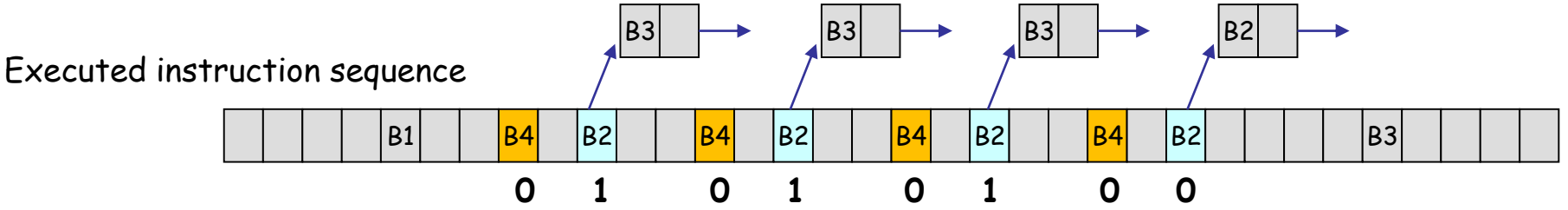
Basic block contains a sequence of statement.
The flow of control enters at the beginning of the statement and leave at the end.

Executed instruction sequence



Predicting the sequence of **01010100** 01010100 01010100 ...

Sample program: vector add with two branches



Predicting the branch outcome sequence

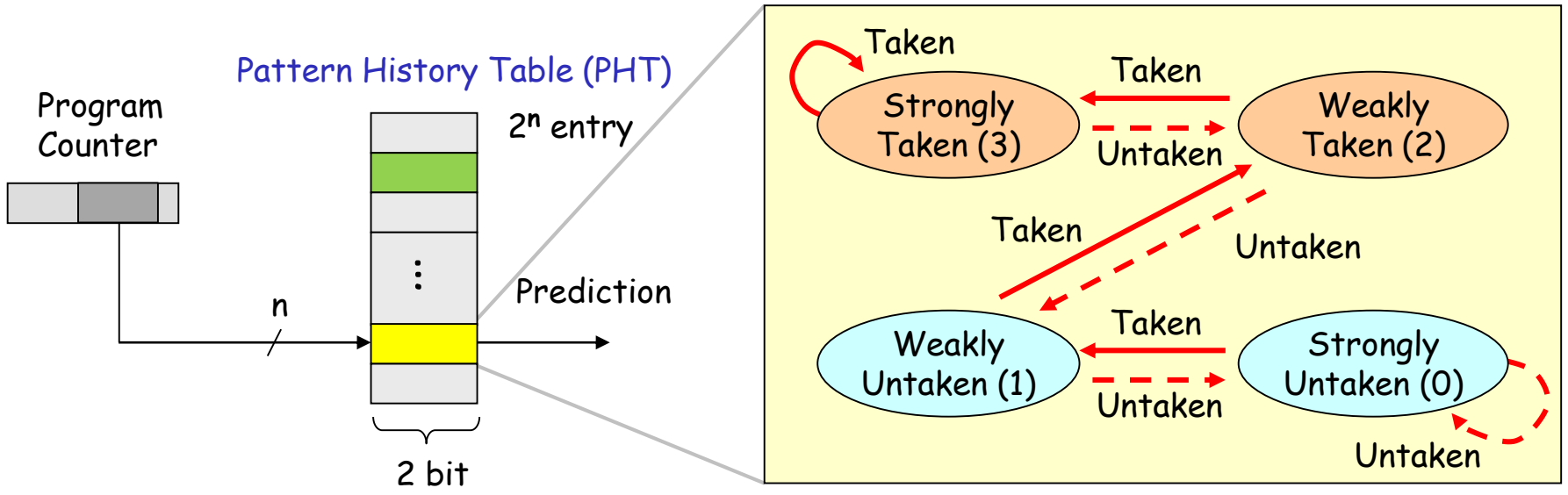
01010100 01010100 01010100 ...

The B4's sequence 01010100 01010100 01010100 ...

The B2's sequence 01010100 01010100 01010100 ...

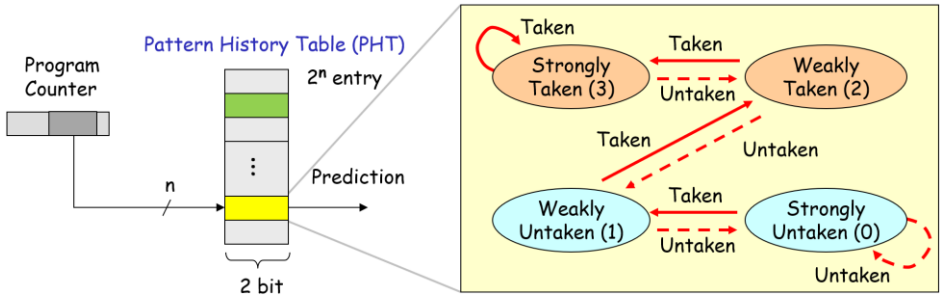
Simple branch predictor: bimodal

- Program has many static branch instructions. The behavior may depend on each branch. Use plenty of counters (PHT) and assign a counter for a branch instruction.
- How to predict
  - Select a 2-bit counter using PC, and it predicts 1 for taken if the MSB of the register is one; otherwise, it predicts 0 for untaken.
- How to update
  - Select a counter using PC, then update the counter in the same way as 2-bit counter.



Simple branch predictor: bimodal

```
#define N 1024    // Number of PHT entries
int pht[N];       // pattern history table
int idx;          // index of PHT
/*****************************************************************************/
void init_predictor()
{
    for(int i=0; i<N; i++) pht[i] = 2;
}

/*****************************************************************************/
int make_prediction(unsigned int pc)
{
    idx = (pc>>2) % N;
    return (pht[idx] & 0x2) ? 1 : 0;
}

/*****************************************************************************/
void train_predictor(unsigned int pc, int outcome)
{
    if(outcome==1 && pht[idx]<3) pht[idx]++;
    if(outcome==0 && pht[idx]>0) pht[idx]--;
}

/*****************************************************************************/
int main()
{
    int pred;     // branch prediction
    int outcome;  // branch outcome (taken/untaken)
    init_predictor();

    int pc = 0x20;
    for(int i=1; i<25; i++) {
        pred = make_prediction(pc); /***** prediction *****/

        outcome = (i % 4) ? 1 : 0; /***** branch outcome: 111011101110... *****/

        printf("%4d: pc=%3x, idx=%d, cnt=%d, pred=%d, outcome=%d ",
               i, pc, idx, pht[idx], pred, outcome);

        train_predictor(pc, outcome); /***** training *****/

        if(pred==outcome) printf("hit\n"); else printf("miss\n");
    }
    return 0;
}
```



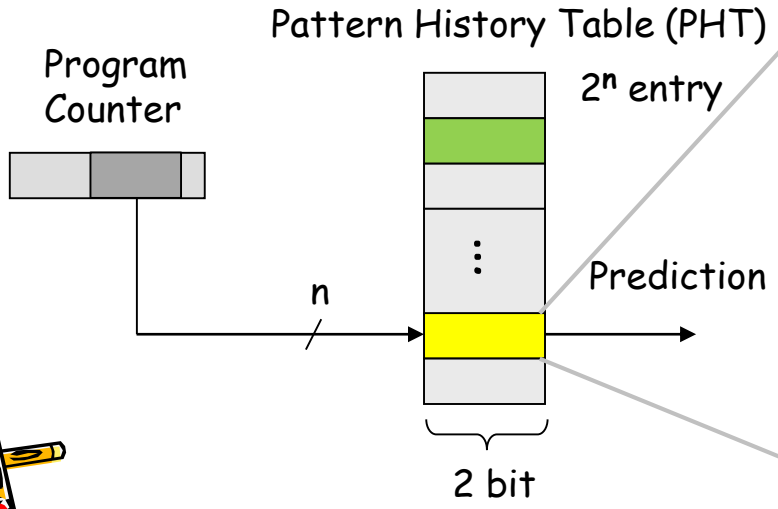
Predicting the branch outcome sequence

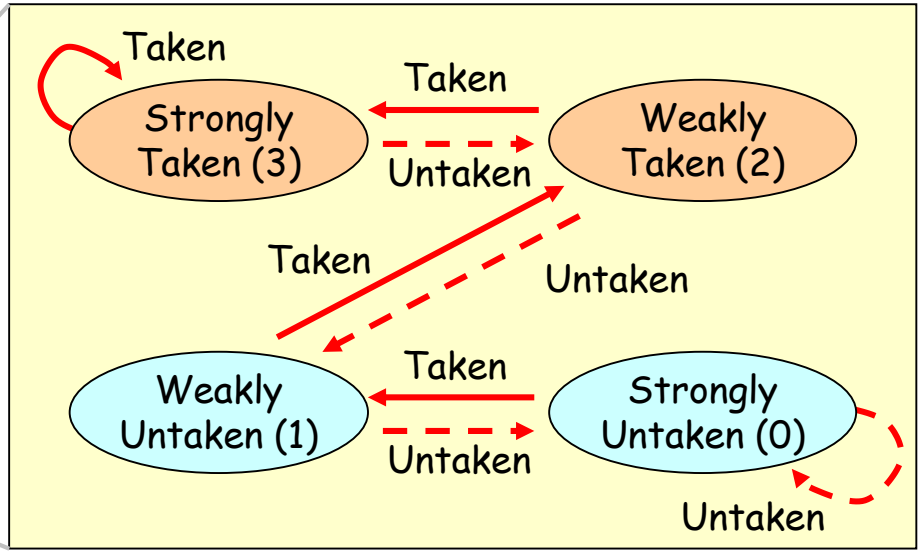
1110 1110 1110 1110 1110 …

```
 1: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
 2: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
 3: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
 4: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
 5: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
 6: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
 7: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
 8: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
 9: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
10: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
11: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
12: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
13: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
14: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
15: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
16: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
17: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
18: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
19: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
20: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
21: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
22: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
23: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
24: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
```

Simple branch predictor: bimodal

Predicting the sequence	01010100 01010100 01010100 ...
The B4's sequence	01010100 01010100 01010100 ...
State of the counter	2 1 0 0 0 0 0 0 0 0 0 0 ...
Prediction	1 0 0 0 0 0 0 0 0 0 0 0 ...
Hit/Miss or the pred.	M H H H H H H H H H H H ...
The B2's sequence	01010100 01010100 01010100 ...
State of the counter	2 3 3 3 2 3 3 3 2 3 3 3 ...
Prediction	1 1 1 1 1 1 1 1 1 1 1 1 ...
Hit/Miss or the pred.	H H H M H H H M H H H M ...





Simple branch predictor: bimodal

```
/************************************************************************************/
int make_prediction(unsigned int pc)
{
    idx = (pc>>2) % N;
    return (pht[idx] & 0x2) ? 1 : 0;
}

/************************************************************************************/
void train_predictor(unsigned int pc, int outcome)
{
    if(outcome==1 && pht[idx]<3) pht[idx]++;
    if(outcome==0 && pht[idx]>0) pht[idx]--;
}

/************************************************************************************/
int main()
{
    int pred;     // branch prediction
    int outcome;  // branch outcome (taken/untaken)
    init_predictor();

    int pc;
    for(int i=1; i<25; i++) {
        if(i&1) { pc = 0x10; } else { pc = 0x20;}

        pred = make_prediction(pc); /***** prediction *****/

        if(pc==0x10) {
            outcome = 0;
        }
        else {
            outcome = (i/2 % 4) ? 1 : 0; /***** outcome: 111011101110... *****/
        }

        printf("%4d: pc=%3x, idx=%d, cnt=%d, pred=%d, outcome=%d ",
               i, pc, idx, pht[idx], pred, outcome);

        train_predictor(pc, outcome); /***** training *****/

        if(pred==outcome) printf("hit\n"); else printf("miss\n");
    }
    return 0;
}
```

Predicting the sequence	0	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	...
The B4's sequence	0	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	...
State of the counter	**2**	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
Prediction	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
Hit/Miss or the pred.	M	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	...
The B2's sequence	0	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	...
State of the counter	**2**	3	3	3	2	3	3	3	2	3	3	3	2	3	3	3	2	3	3	3	...
Prediction	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
Hit/Miss or the pred.	H	H	H	M	H	H	H	M	H	H	H	M	H	H	H	M	H	H	H	M	...

```
   1: pc= 10, idx=4, cnt=2, pred=1, outcome=0 miss
   2: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
   3: pc= 10, idx=4, cnt=1, pred=0, outcome=0 hit
   4: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
   5: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
   6: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
   7: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
   8: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
   9: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  10: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
  11: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  12: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
  13: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  14: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
  15: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  16: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
  17: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  18: pc= 20, idx=8, cnt=2, pred=1, outcome=1 hit
  19: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  20: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
  21: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  22: pc= 20, idx=8, cnt=3, pred=1, outcome=1 hit
  23: pc= 10, idx=4, cnt=0, pred=0, outcome=0 hit
  24: pc= 20, idx=8, cnt=3, pred=1, outcome=0 miss
```

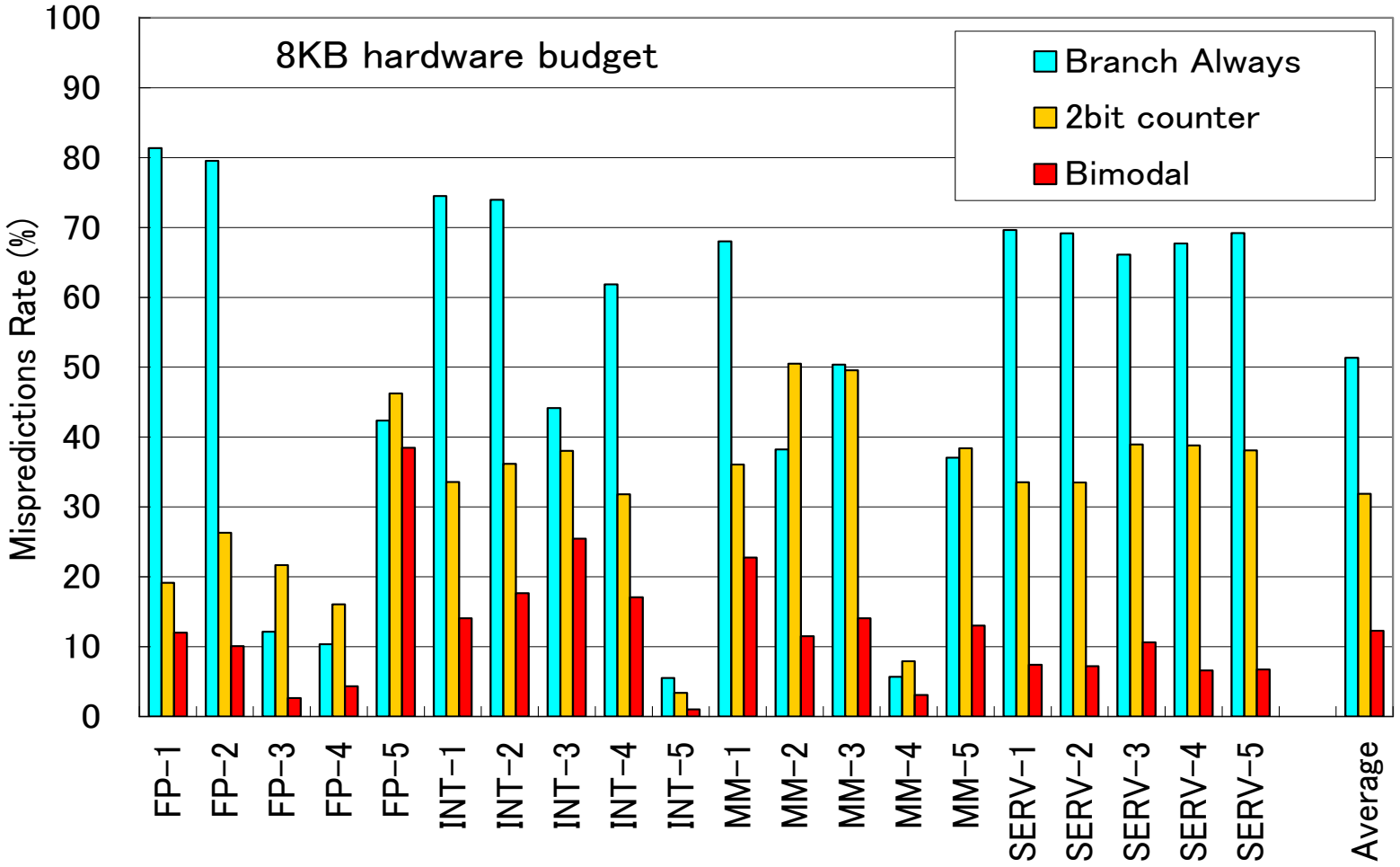
branch predictor in C : bimodal

```c
/** BPkit: bimodal branch predictor version 1.0                          **/
/** 2024-10-24  Kenji KISE, ArchLab, Science Tokyo                       **/
/** Released under the MIT license https://opensource.org/licenses/mit **/
/**************************************************************************/
#include <stdio.h>
#include <stdlib.h>

#define PHT_SIZE (16*1024)  /* the number of PHT entries    */
#define BTB_SIZE ( 2*1024)  /* the number of BTB entries    */
int pht[PHT_SIZE];          /* PHT, pattern history table   */
int btb[BTB_SIZE];          /* BTB, branch target buffer    */
/**************************************************************************/
void bp_init(){
    for(int i=0; i<PHT_SIZE; i++) pht[i] = 1; // weak untaken
    for(int i=0; i<BTB_SIZE; i++) btb[i] = 0;
}

/**************************************************************************/
void bp_predict(unsigned int pc, int *pred, unsigned int *p_target){
    int btb_idx = (pc>>2) % BTB_SIZE;
    *p_target = btb[btb_idx];

    int pht_idx = (pc>>2) % PHT_SIZE;
    *pred = (btb[btb_idx]!=0 && (pht[pht_idx] >> 1));
}

/**************************************************************************/
void bp_regist(unsigned int pc, int taken, unsigned int target){
    int btb_idx = (pc>>2) % BTB_SIZE;
    btb[btb_idx] = target;

    int pht_idx = (pc>>2) % PHT_SIZE;
    pht[pht_idx] = (taken) ? pht[pht_idx] + (pht[pht_idx]<3) :
                             pht[pht_idx] - (pht[pht_idx]>0);
}
```

```c
/**************************************************************************/
int main(int argc, char **argv){
    int count=0, hit=0, mis=0;
    FILE *fp;

    if(argc==1){
        printf("usage: %s tracef_filename\n", argv[0]);
        exit(1);
    }
    if((fp = fopen(argv[1], "r"))==NULL){
        printf("Bad file name: %s\n", argv[1]);
        exit(0);
    }

    bp_init();

    while(1){
        int pred, taken, p_mis;
        unsigned int pc, target, p_target;
        int ret = fscanf(fp, "%x %x %d", &pc, &target, &taken);
        if(ret==EOF) break;

        bp_predict(pc, &pred, &p_target);

        p_mis = (pred!=taken || (taken==1 && target!=p_target));
        if (p_mis) mis++; else hit++;

        bp_regist(pc, taken, target);
        count++;
    }

    float m_rate = (float)mis/(float)(hit+mis);
    printf("=> the number of branch predictions    : %10d\n",   count);
    printf("=> the number of branch mispredictions : %10d\n",   mis);
    printf("=> branch prediction miss rate         : %10.4f\n", m_rate);
    return 0;
}
/**************************************************************************/
```

bimodal.c

Accuracy of simple predictors with 8KB HW budget



Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.

An innovation in branch predictors in 1993

- Using branch history
  - global branch history
  - local branch history
- 2-level branch predictor and gshare
- Assume predicting the sequence 1110 1110 1110 1110 1110 ...

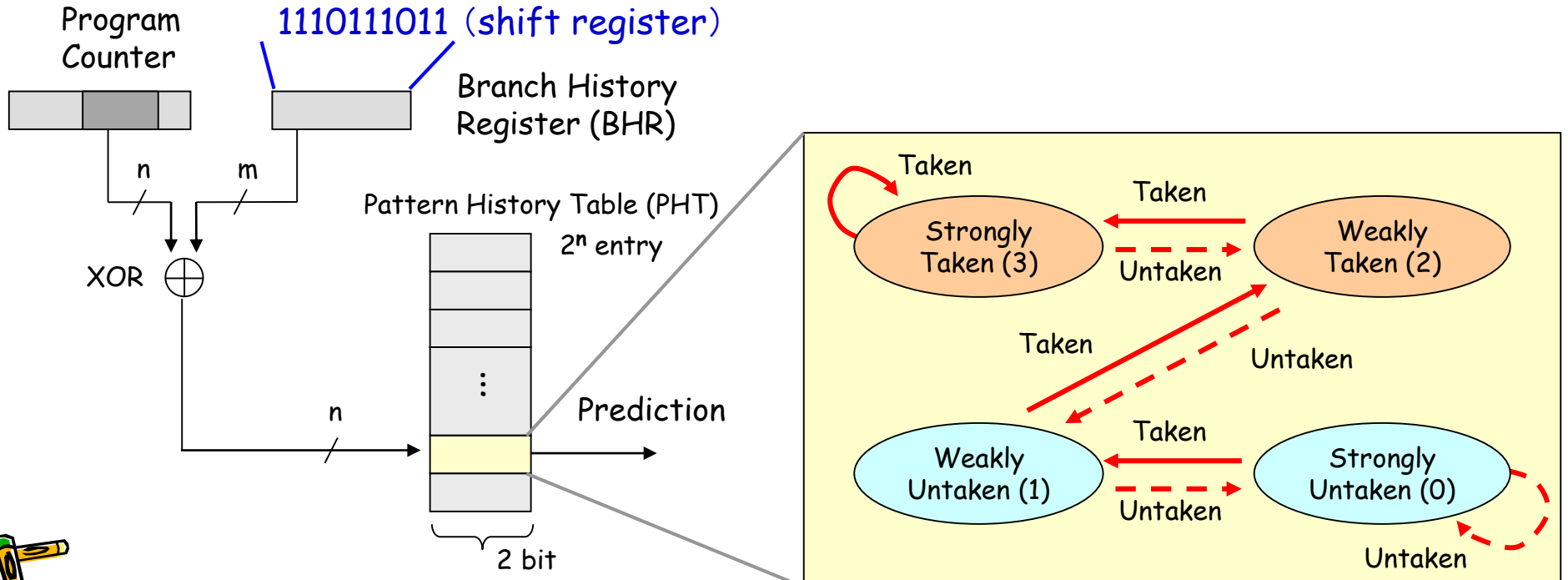
```
1110111 0
11101110 ?
111011101 ?
1110111011 ?
11101110111 ?
111011101110 ?
```

adr	pred
000	
001	
010	
011	1
100	
101	1
110	1
111	0

Use the recent branch history as an address of a table.

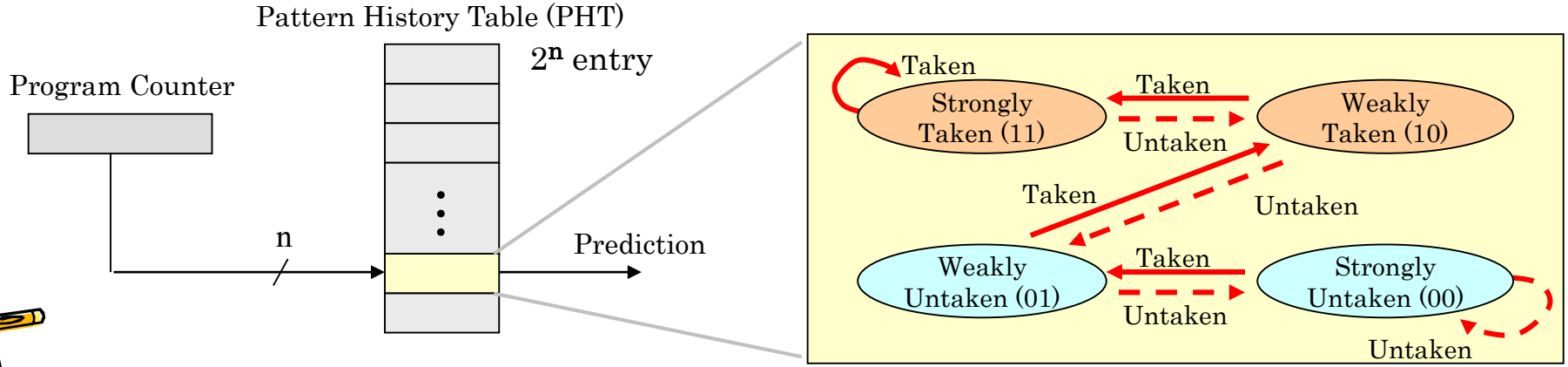
Gshare (TR-DEC 1993)

- How to predict
  - Using the exclusive OR of the global branch history and PC to access PHT, then MSB of the selected counter is the prediction.
- How to update
  - Shifting BHR one bit left and update LSB by branch outcome in IF stage.
  - Update the used counter in the same way as 2BC in WB stage.



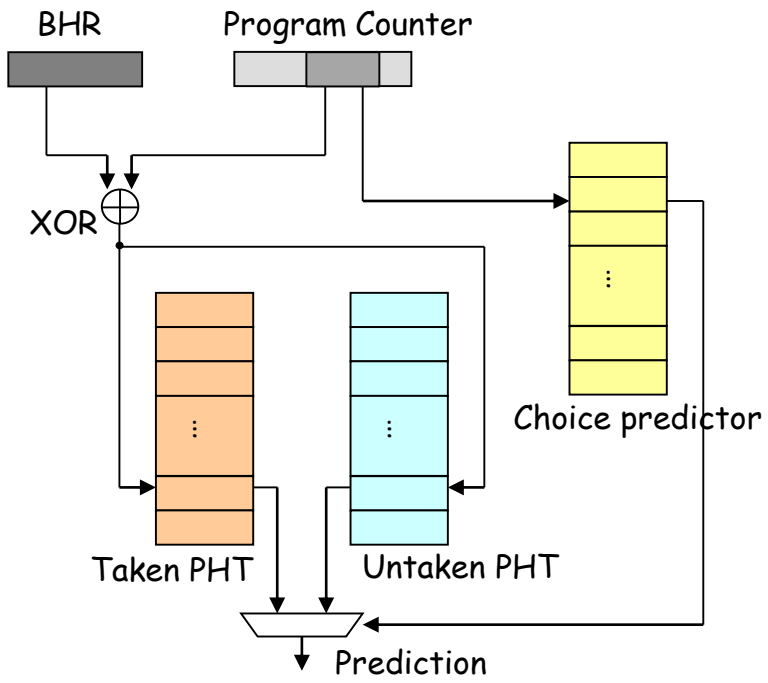
Gshare branch predictor

```
 1  module m_gshare (
 2    input  wire w_clk, w_we,
 3    input  wire [4:0] w_wadr, w_radr,
 4    input  wire w_tkn,
 5    input  wire [4:0] w_r_brh, w_w_bhr,
 6    output wire w_pred
 7  );
 8    reg [1:0] mem [0:31];
 9    wire [1:0] w_data = mem[w_radr ^ w_r_bhr];
10    assign w_pred = w_data[1];
11
12    wire [1:0] w_cnt  = mem[w_wadr ^ w_w_bhr];
13    always @(posedge w_clk) if (w_we)
14      mem[w_wadr ^ w_w_bhr] <= (w_cnt < 3 &  w_tkn) ? w_cnt + 1 :
15                               (w_cnt > 0 & !w_tkn) ? w_cnt - 1 : w_cnt;
16    integer i; initial for (i=0; i<32; i=i+1) mem[i] = 1;
17  endmodule
```



Bi-Mode (MICRO 1997)

- A choice predictor (bimodal) is used as a meta-predictor
- How to predict
  - Like gshare, both of Taken PHT and Untaken PHT make two predictions.
  - Select one among them by the choice predictor which tracks the global bias of a branch.
- How to update
  - The used PHT is updated in the same way as 2BC.
  - Choice predictor is updated in the same way as bimodal.



To go beyond gshare

- Using branch history
  - global branch history
  - local branch history
- 2-level branch predictor and gshare

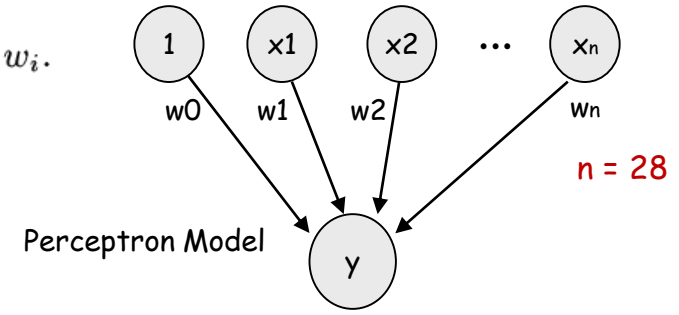
- Assume predicting the sequence 1110 1110 1110 1110 1110 …

```
11101110 ?          11101110 ?
111011101 ?         111011101 ?
1110111011 ?        1110111011 ?
11101110111 ?       11101110111 ?
1110111101110 ?     1110111101110 ?
```

Perceptron (HPCA 2001)

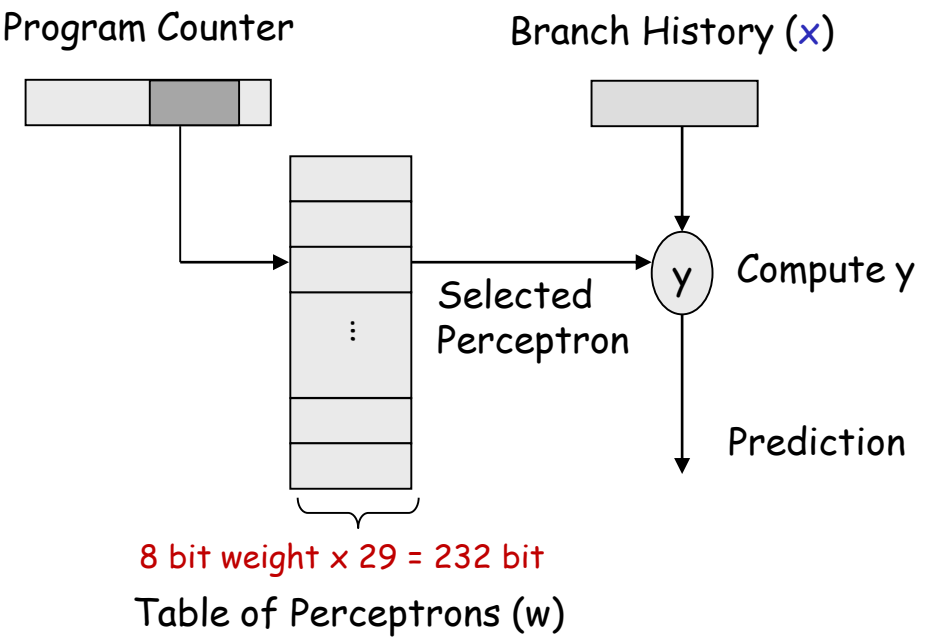
- How to predict
  - Select one perceptron by PC
  - Compute y using the equation. It predicts 1 if y>=0, predicts 0 if y<0
  - x is branch history. xi is either -1, meaning not taken or 1, meaning taken
- How to update
  - Train the weights of used perceptron when the prediction miss or |y| < T (Threshold)

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



$$\begin{aligned}
&\texttt{if } \mathrm{sign}(y_{out}) \neq t \texttt{ or } |y_{out}| \leq \theta \texttt{ then} \\
&\quad \texttt{for } i := 0 \texttt{ to } n \texttt{ do} \\
&\quad\quad w_i := w_i + t x_i \\
&\quad \texttt{end for} \\
&\texttt{end if}
\end{aligned}$$

T = 1.93n + 14



Perceptron (HPCA 2001)

- How to predict
  - Select one perceptron by PC
  - Compute y using the equation. It predicts 1 if y>=0, predicts 0 if y<0
  - x is branch history. xi is either -1, meaning not taken or 1, meaning taken
- How to update
  - Train the weights of used perceptron when the prediction miss or |y| < T (Threshold)

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

```
Number of weights (without bias) of perceptron: 4
Theta:  21.720
   1: Wn-W0 =   0   0   0   0   0 : bhr=0000: y=  0, p=1 : out=1 : hit
   2: Wn-W0 =  -1  -1  -1  -1   1 : bhr=0001: y=  3, p=1 : out=1 : hit
   3: Wn-W0 =  -2  -2  -2   0   2 : bhr=0011: y=  4, p=1 : out=1 : hit
   4: Wn-W0 =  -3  -3  -1   1   3 : bhr=0111: y=  3, p=1 : out=0 : miss
   5: Wn-W0 =  -2  -4  -2   0   2 : bhr=1110: y= -6, p=0 : out=1 : miss
   6: Wn-W0 =  -1  -3  -1  -1   3 : bhr=1101: y= -1, p=0 : out=1 : miss
   7: Wn-W0 =   0  -2  -2   0   4 : bhr=1011: y=  4, p=1 : out=1 : hit
   8: Wn-W0 =   1  -3  -1   1   5 : bhr=0111: y=  1, p=1 : out=0 : miss
   9: Wn-W0 =   2  -4  -2   0   4 : bhr=1110: y=  0, p=1 : out=1 : hit
  10: Wn-W0 =   3  -3  -1  -1   5 : bhr=1101: y=  5, p=1 : out=1 : hit
  11: Wn-W0 =   4  -2  -2   0   6 : bhr=1011: y= 10, p=1 : out=1 : hit
  12: Wn-W0 =   5  -3  -1   1   7 : bhr=0111: y= -1, p=0 : out=0 : hit
  13: Wn-W0 =   6  -4  -2   0   6 : bhr=1110: y=  6, p=1 : out=1 : hit
  14: Wn-W0 =   7  -3  -1  -1   7 : bhr=1101: y= 11, p=1 : out=1 : hit
  15: Wn-W0 =   8  -2  -2   0   8 : bhr=1011: y= 16, p=1 : out=1 : hit
  16: Wn-W0 =   9  -3  -1   1   9 : bhr=0111: y= -3, p=0 : out=0 : hit
  17: Wn-W0 =  10  -4  -2   0   8 : bhr=1110: y= 12, p=1 : out=1 : hit
  18: Wn-W0 =  11  -3  -1  -1   9 : bhr=1101: y= 17, p=1 : out=1 : hit
  19: Wn-W0 =  12  -2  -2   0  10 : bhr=1011: y= 22, p=1 : out=1 : hit
  20: Wn-W0 =  12  -2  -2   0  10 : bhr=0111: y= -6, p=0 : out=0 : hit
  21: Wn-W0 =  13  -3  -3  -1   9 : bhr=1110: y= 17, p=1 : out=1 : hit
  22: Wn-W0 =  14  -2  -2  -2  10 : bhr=1101: y= 22, p=1 : out=1 : hit
  23: Wn-W0 =  14  -2  -2  -2  10 : bhr=1011: y= 22, p=1 : out=1 : hit
  24: Wn-W0 =  14  -2  -2  -2  10 : bhr=0111: y=-10, p=0 : out=0 : hit
  25: Wn-W0 =  15  -3  -3  -3   9 : bhr=1110: y= 21, p=1 : out=1 : hit
  26: Wn-W0 =  16  -2  -2  -4  10 : bhr=1101: y= 22, p=1 : out=1 : hit
  27: Wn-W0 =  16  -2  -2  -4  10 : bhr=1011: y= 22, p=1 : out=1 : hit
  28: Wn-W0 =  16  -2  -2  -4  10 : bhr=0111: y=-14, p=0 : out=0 : hit
  29: Wn-W0 =  17  -3  -3  -5   9 : bhr=1110: y= 25, p=1 : out=1 : hit
```

```
if sign(y_out) ≠ t or |y_out| ≤ θ then
        for i := 0 to n do
                w_i := w_i + t x_i
        end for
end if
```

T = 1.93n + 14

Perceptron (HPCA 2001)

```
/*****************************************************************************/
/* perceptron based branch predictor Version v2024-12-26a                    */
/*  Copyright (c) 2024 Archlab. Science Tokyo                                */
/*  Released under the MIT license https://opensource.org/licenses/mit       */
/*****************************************************************************/
#include <stdio.h>

#define N 4                       // Number of weights of perceptron, default 28
#define BitsInWeight 8            // Number of bits in a weight
#define MAXVAL  127               // max value of a weight
#define MINVAL -128               // min value of a weight
#define NPerceptron (1024)        // the number of perceptrons
#define ThetaMax (N * 1.93 + 14)  // Threshold max value
#define ThetaMin (-1 * ThetaMax)  // Threshold min value

int perceptron[NPerceptron][N+1]; // perceptron table
int bhr;                          // global branch history register
int idx;                          // index of perceptron table
int y;                            // weighted sum with bias
int prediction;                   // prediction of taken/untaken

/*****************************************************************************/
void init_predictor()
{
    for(int i=0; i<NPerceptron; i++){
        for(int j=0 ; j<=N ; j++){
            perceptron[i][j] = 0;
        }
    }
    bhr = 0;
}

/*****************************************************************************/
int make_prediction(unsigned int pc)
{
    idx = (pc>>2) % NPerceptron;

    y = perceptron[idx][0];
    for(int i=1; i<=N; i++){
        if((bhr >> (i-1)) & 1) y += perceptron[idx][i];
        else                   y -= perceptron[idx][i];
    }

    prediction = (y >= 0) ? 1 : 0;
    return prediction;
}
```

```
void train_predictor(unsigned int pc, int outcome)
{
    if(outcome != prediction || ((y < ThetaMax) && (y > ThetaMin))){

        int *bias = &perceptron[idx][0];
        if(outcome==1 && (*bias < MAXVAL)) *bias = *bias + 1;
        if(outcome==0 && (*bias > MINVAL)) *bias = *bias - 1;

        for(int i=1; i <=N; i++){
            if(((bhr >> (i-1)) & 1)==outcome){
                if (perceptron[idx][i] < MAXVAL) perceptron[idx][i]++;
            }
            else{
                if (perceptron[idx][i] > MINVAL) perceptron[idx][i]--;
            }
        }
    }
    bhr = (bhr << 1) | outcome;
}

/*****************************************************************************/
int main()
{
    int pred;     // branch prediction
    int outcome;  // branch outcome (taken/untaken)
    init_predictor();
    printf("Number of weights (without bias) of perceptron: %d\n", N);
    printf("Theta: %7.3f\n", ThetaMax);

    int pc = 0x2000;
    for(int i=1; i<30; i++) {
        pred = make_prediction(pc); /***** prediction *****/

        printf("%4d: Wn-W0 = ", i);
        for(int i=N; i>=0; i--) printf("%3d ", perceptron[idx][i]);

        outcome = (i % 4) ? 1 : 0; /***** branch outcome: 111011101110... *****/

        printf(": bhr=");
        for(int j=N-1; j>=0; j--){
            printf("%d", ((bhr>>j) & 1));
        }
        printf(": y=%3d, p=%d : out=%d : ", y, pred, outcome);

        train_predictor(pc, outcome); /***** training *****/


        if(pred==outcome) printf("hit\n"); else printf("miss\n");
    }
    return 0;
}
```

Perceptron (HPCA 2001)

The Neural Network in Your CPU

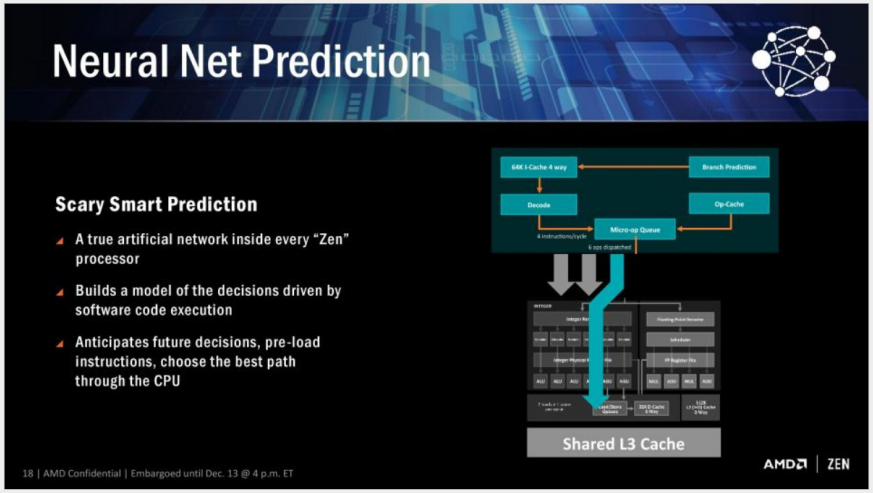
Sun, Aug 6, 2017

Machine learning and artificial intelligence are the current hype (again). In their new Ryzen processors, AMD advertises the Neural Net Prediction. It turns out this is was already used in their older (2012) Piledriver architecture used for example in the AMD A10-4600M. It is also present in recent Samsung processors such as the one powering the Galaxy S7. What is it really?

The basic idea can be traced to a paper from Daniel Jimenez and Calvin Lin "Dynamic Branch Prediction with Perceptrons", more precisely described in the subsequent paper "Neural methods for dynamic branch prediction". Branches typically occur in `if-then-else` statements. Branch prediction consists in guessing which code branch, the `then` or the `else`, the code will execute, thus allowing to precompute the branch in parallel for faster evaluation.

Jimenez and Lin rely on a simple single-layer perceptron neural network whose input are the branch outcome (global or hybrid local and global) histories and the output predicts which branch will be taken. In reality, because there is a single layer,

AMD Ryzen 2016-12-13 Slide Deck — Back to Post

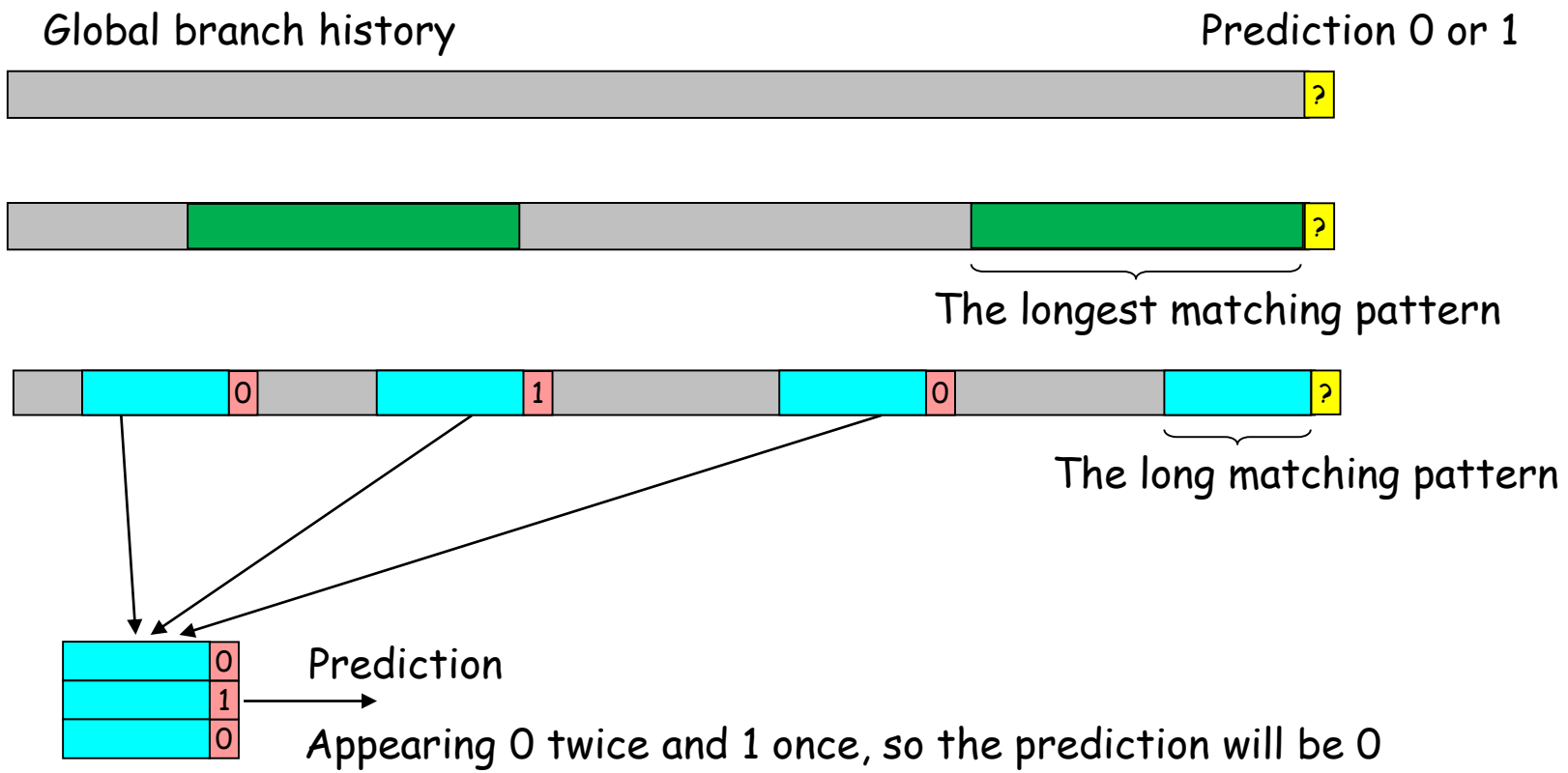


https://www.anandtech.com/Gallery/Album/5197#18

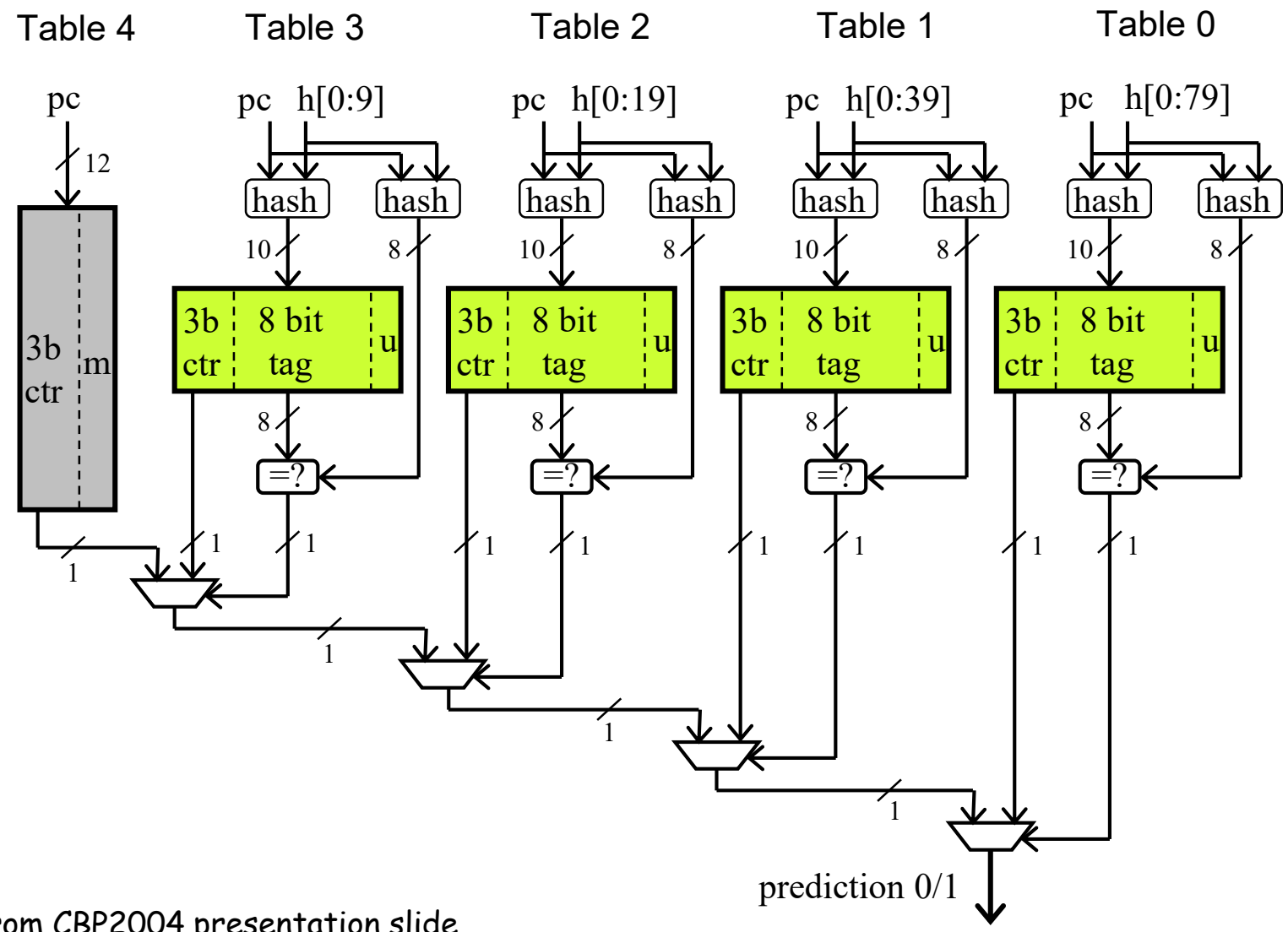
https://chasethedevil.github.io/post/the_neural_network_in_your_cpu/

Branch predictors based on pattern matching

- Find the longest matching pattern (green rectangle)
- Select the proper matching length or long matching pattern (blue rectangle)
- Count the number of 0 and the number of 1 after the long matting patterns (red rectangle), then predict by majority vote.

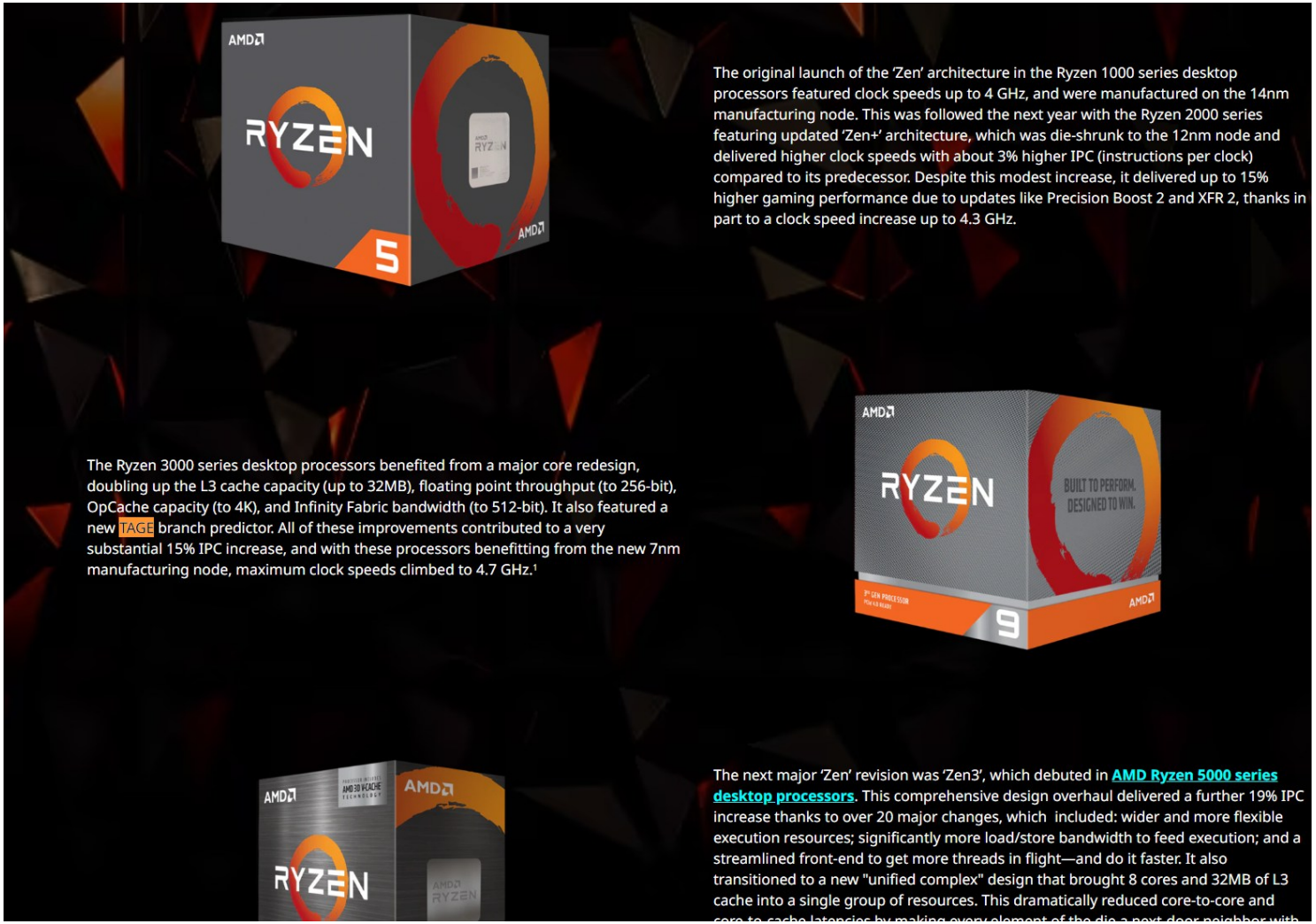


Partial Pattern Matching, PPM or TAGE (CBP 2004)



From CBP2004 presentation slide

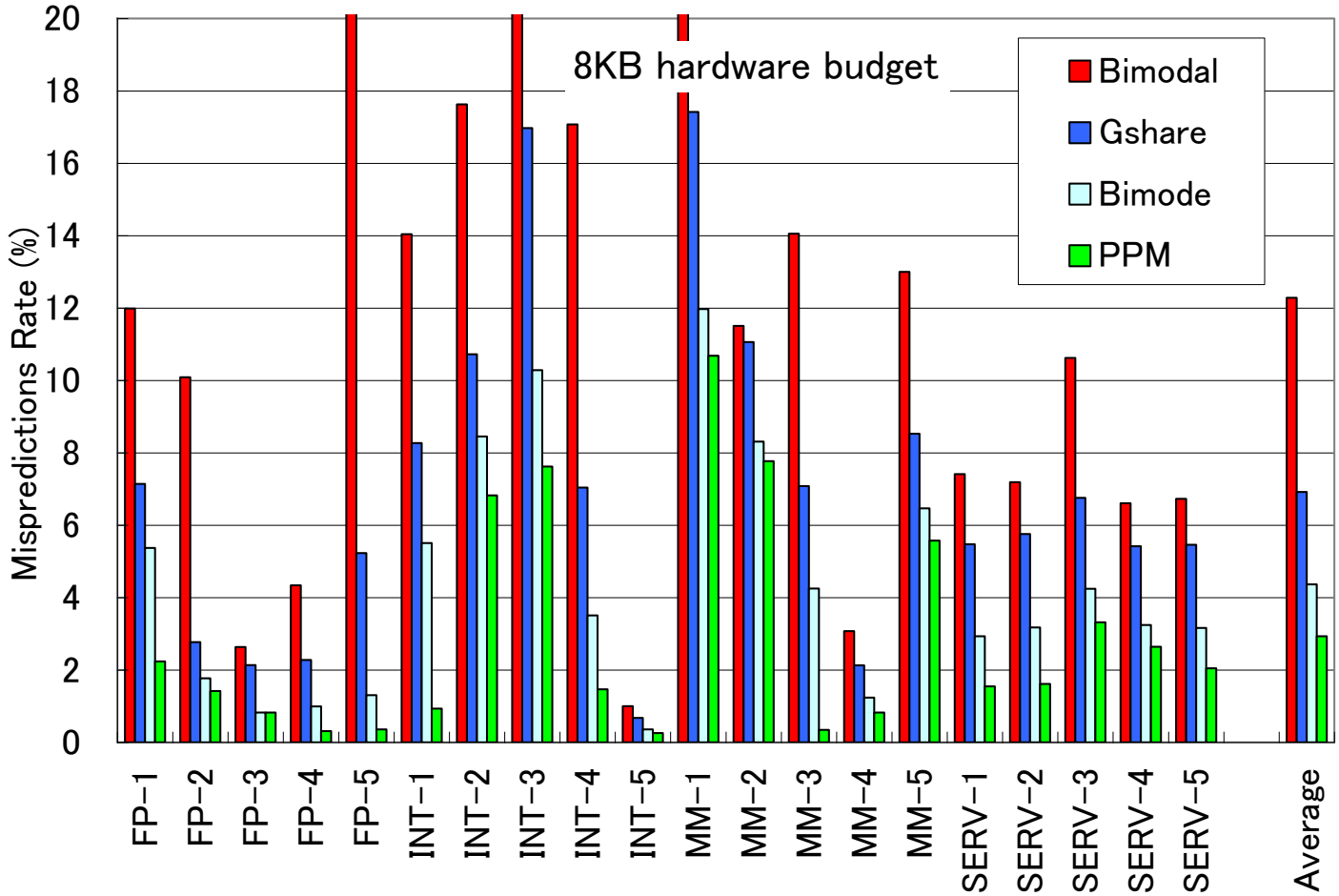
Partial Pattern Matching, PPM or TAGE (CBP 2004)



https://www.amd.com/en/technologies/zen-core

Prediction accuracy

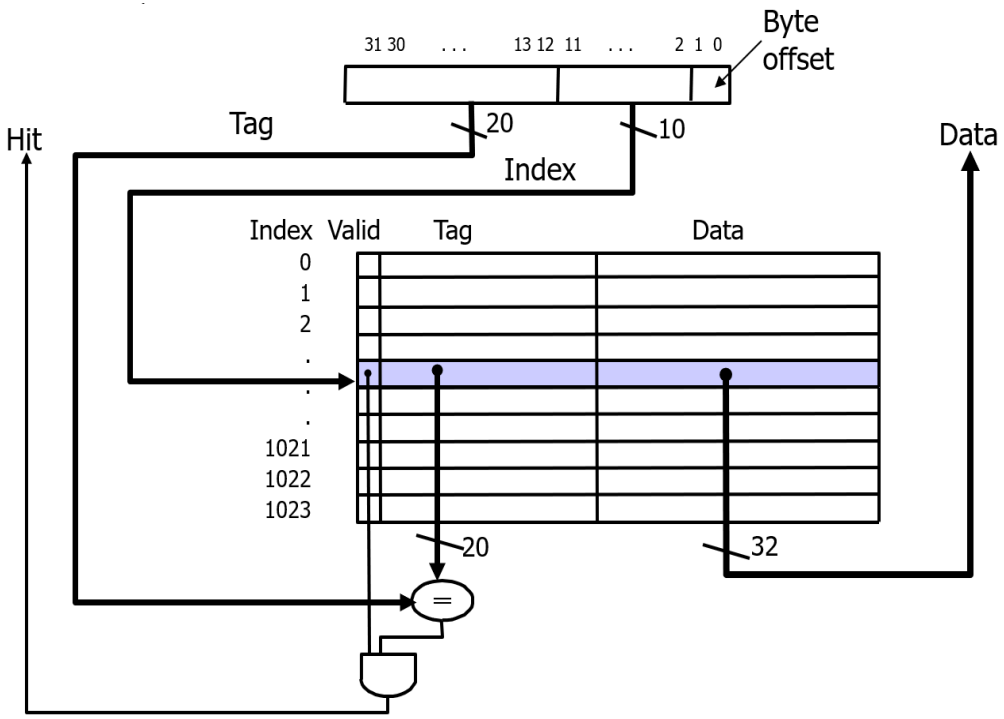
- The accuracy of 4KB Gshare is about 93%.
- The accuracy of 4KB PPM is about 97%.



BTB (Branch Target Buffer)

- フェッチしている命令が分岐命令か？
- 分岐先アドレス(32bit)

Direct Mapped Cache Example




```
module m_btb (
  input  wire        w_clk, w_we,
  input  wire [31:0] w_adr,
  input  wire [4:0]  w_wadr,
  input  wire [57:0] w_wd,
  output wire        w_hit,
  output wire [31:0] w_dout
);
  wire w_v;
  wire [24:0] w_tag;
  reg [57:0] mem [0:31];
  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
  assign {w_v, w_tag, w_dout} = mem[w_adr[6:2]];
  assign w_hit = w_v & (w_tag==w_adr[31:7]);
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;
endmodule
```

Recommended Reading

- Combining Branch Predictors
  - Scott McFarling, Digital Western Research Laboratory
  - WRL Technical Note TN-36, 1993

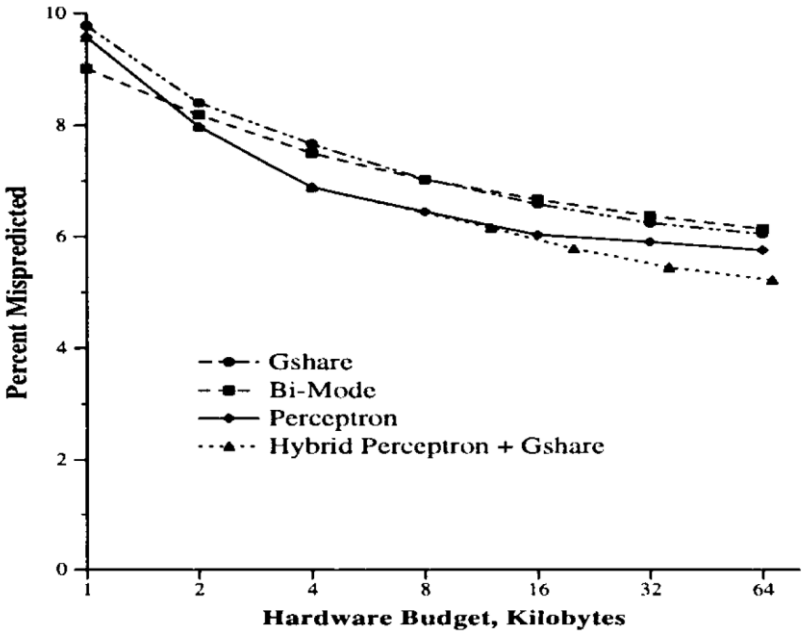
- *A quote:*
  "In this paper, we have presented two new methods for improving branch prediction performance. First, we showed that using the bit-wise exclusive OR of the global branch history and the branch address to access predictor counters results in better performance for a given counter array size."

Recommended Reading

- Dynamic branch prediction with perceptrons
  - Daniel A. Jimenez, Calvin Lin (The University of Texas at Austin)
  - HPCA-7, pp. 197-206 (2001)

Hardware budget in kilobytes	History Length		
	gshare	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.



Perceptron vs. other techniqes, Harmonic Mean

Figure 3: Hardware Budget vs. Prediction Rate on SPEC 2000. The perceptron predictor is more accurate than the two PHT methods at all hardware budgets over one kilobyte.