Computer Architecture

(Fall 2020)

转冰控制冒险

Dynamic Hardware Branch Prediction

& Branch-Target Buffers

Dr. Chunhua Xiao (肖春华)

Office: Main Building 0602

Email: xiaochunhua@cqu.edu.cn

预习 C3, 附录

Review: Reducing CPI (or Increasing IPC)

$$CPI = CPI_{ideal} + stalls_{structural} + stalls_{dataHazard} + stalls_{control}$$

technique	reduces
forwarding/ bypassing	potential data-hazard stalls
delayed branches	control-hazard stalls
basic dynamic scheduling (scoreboarding)	data-hazard stalls from true dependencies
dynamic scheduling with register renaming	data-hazard, anti-dep. & output dep. stalls
dynamic branch prediction	control stalls
issuing multiple instruction per clock cycle	ideal CPI
speculation	data-hazard and control-hazard stalls
dynamic memory disambiguation	data-hazard stalls with memory
loop unrolling	control hazard stalls
basic compiler pipeline scheduling	data-hazard stalls
compiler dependency analysis	ideal CPI, data-hazard stalls
software pipelining & trace scheduling	ideal CPI, data-hazard stalls
compiler speculation	ideal CPI, data-hazard stalls

Reducing Branch Penalty

Branch penalty : wasted cycles due to pipeline flushing on mis-predicted branches

Reduce branch penalty:

- Predict branch/jump instructions AND branch direction (taken or not taken)
- Predict branch/jump target address (for taken branches)
- Speculatively execute instructions along the predicted path

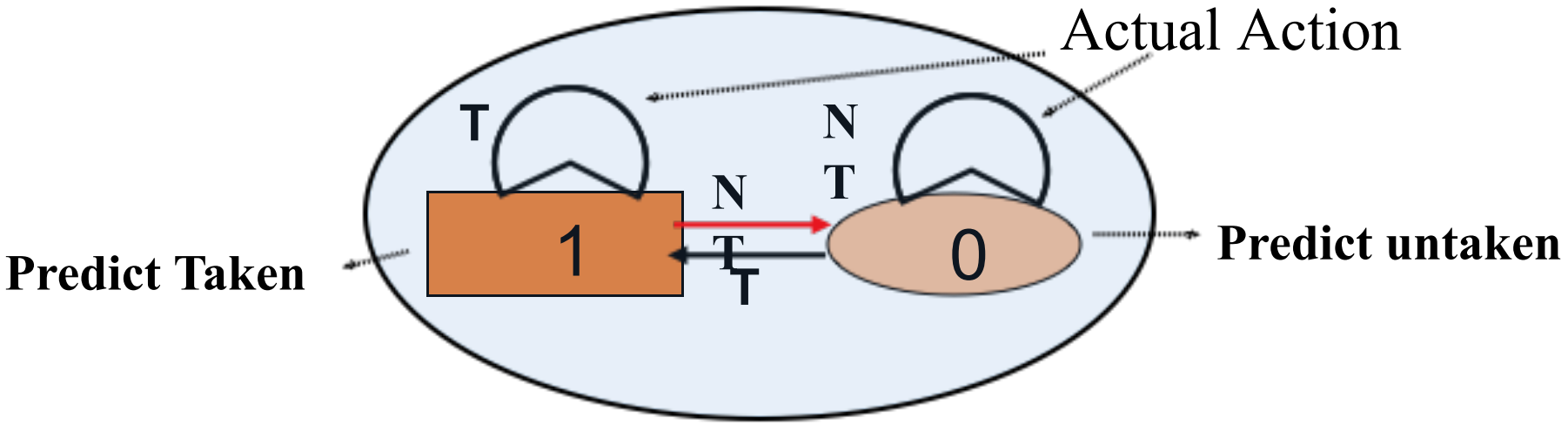
Branch Prediction Strategies

策略

- Static
- Decided before runtime
- Examples:
- Always-Not Taken 预测
- Always-Taken
- Backwards Taken, Forward Not Taken (BTFNT)
- Profile-driven prediction 历史信息
- Dynamic
- Prediction decisions may change during the execution of the program

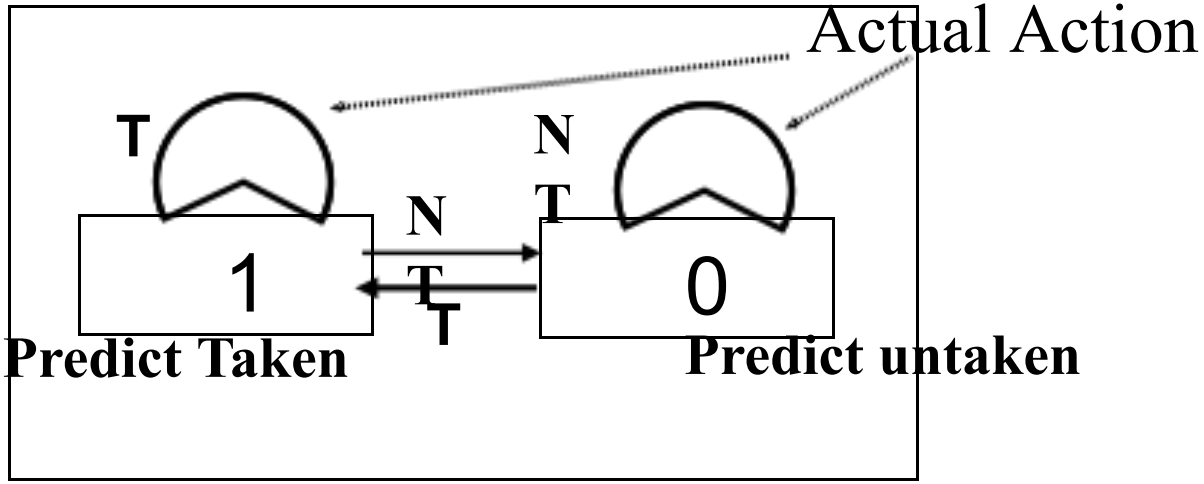
A Simple Scheme: 1-bit Predictor for a Single Branch

动态调整



1-bit prediction

A Simple Scheme: 1-bit Predictor for a Single Branch



1-bit prediction

Basic Branch Prediction: Branch-History Table (or Branch-Prediction Buffer)

Implemented as a small memory indexed by a portion (usually some low-significant bits) of the address of the branch instruction.

So the size of this table is the number of entries □ the number of bit per entry.

If unfortunately, the address portions of two branch instructions are identical, they share one entry. Hence, the more entries, the less such conflicts.

e.g., branch instructions at the following addresses share the BHT entry

- 00F2BC 0101 1100
- 010A5D 1001 1100

Index	Taken?
0000	1
0001	0
0010	1
0011	0
0100	1
0101	0
0110	1
0111	1
1000	1
1001	0
1010	1
1011	1
1100	0
1101	1
1110	1
1111	1

1-bit Predictor Weakness on nested loops

- Consider a nested loop :

```
for (…){
  for (i=0; i<10; i++)
     a[i] = a[i] * 2.0;
}
```

- Mispredict twice, once on entry, once on exit, every time when the inner loop is executed.

Example: Accuracy of 1-Bit Prediction Scheme in the Presence of "Loop Branching"

```
loop: L.D   F0, 0(R1)
      MUL.D F4, F0, F2
      S.D   F4, 0(R1)
      DADDIU R1, R1, #-8
      BNEZ  R1 loop
```

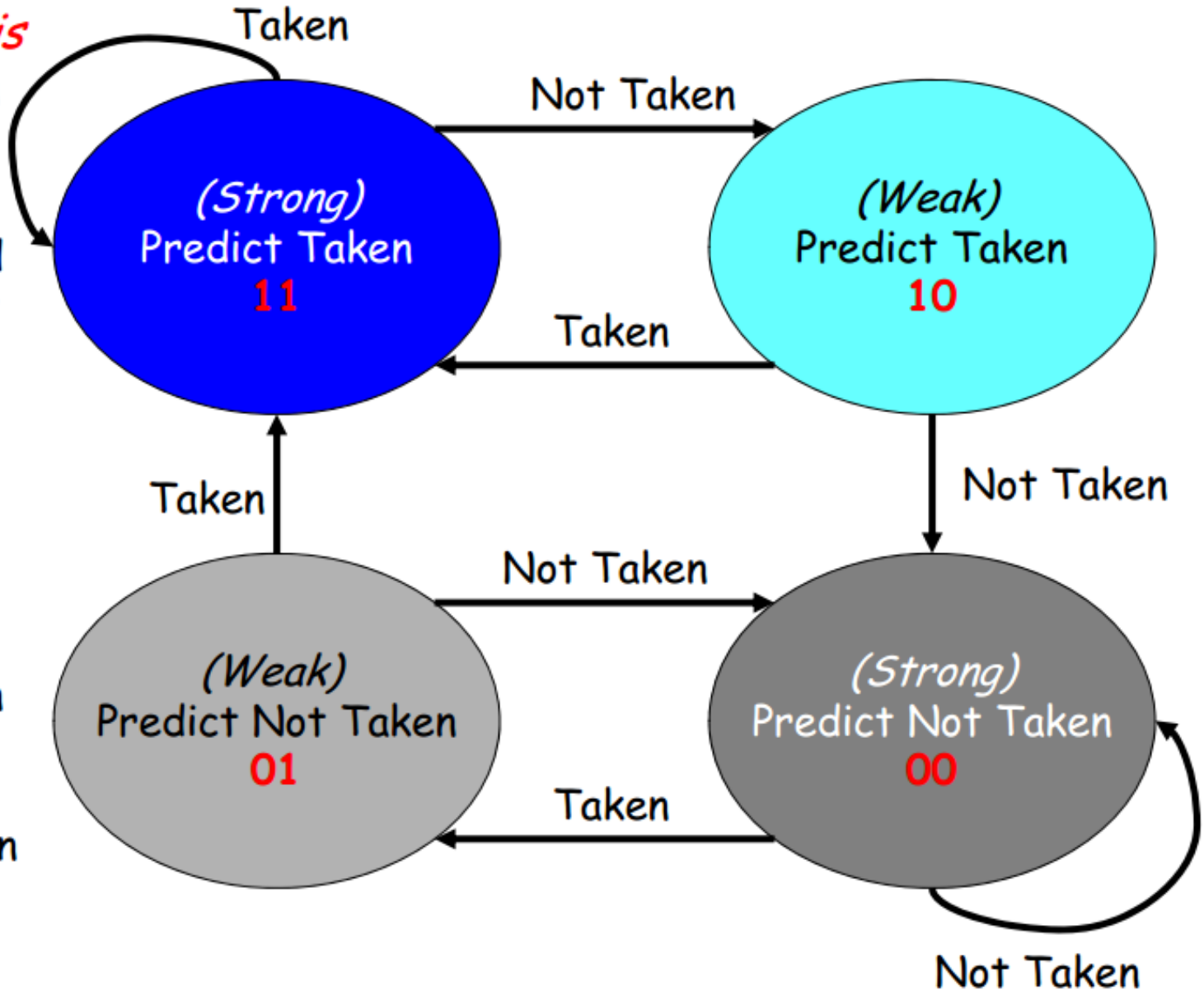
- Assumptions
  - R1 is initialized to #80

Iteration	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4	5	6	7	8	9	10
Predicted behavior	NT	T	T	T	T	T	T	T	T	T	T	NT	T	T	T	T	T	T	T	T	T	T
Actual behavior	T	T	T	T	T	T	T	T	T	T	NT	T	T	T	T	T	T	T	T	T	T	NT

- Branch taken 90% of the times, but branch prediction accuracy is only 80%
- A 1-bit predictor for "loop branches" mispredicts at twice the rate that the branch is not taken

2-Bit Prediction Scheme (a.k.a. Bimodal Predictor)

- Adding *hysteresis* to the prediction scheme
  - a prediction must be missed twice before it is changed
- Implementation
  - a *(2-bit) saturated counter* that is incremented on a taken branch and decremented on an untaken branch



Example: 2-Bit Prediction Scheme in Action with "Loop Branching"

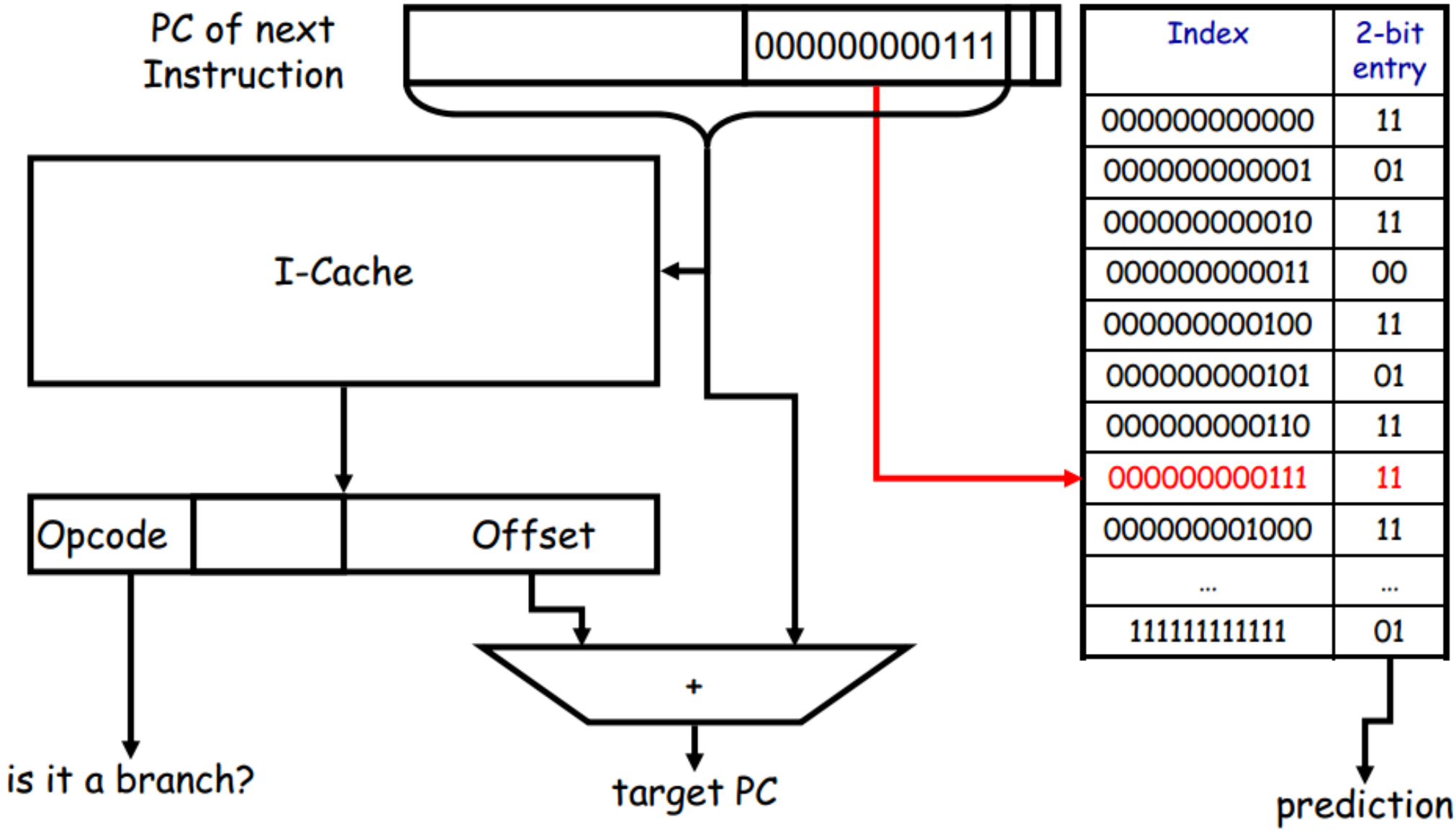
```
loop: L.D F0, 0(R1)
      MUL.D F4, F0, F2
      S.D F4, 0(R1)
      DADDIU R1, R1, #-8
      BNEZ R1 loop
```

- Assumptions
  - R1 is initialized to #80

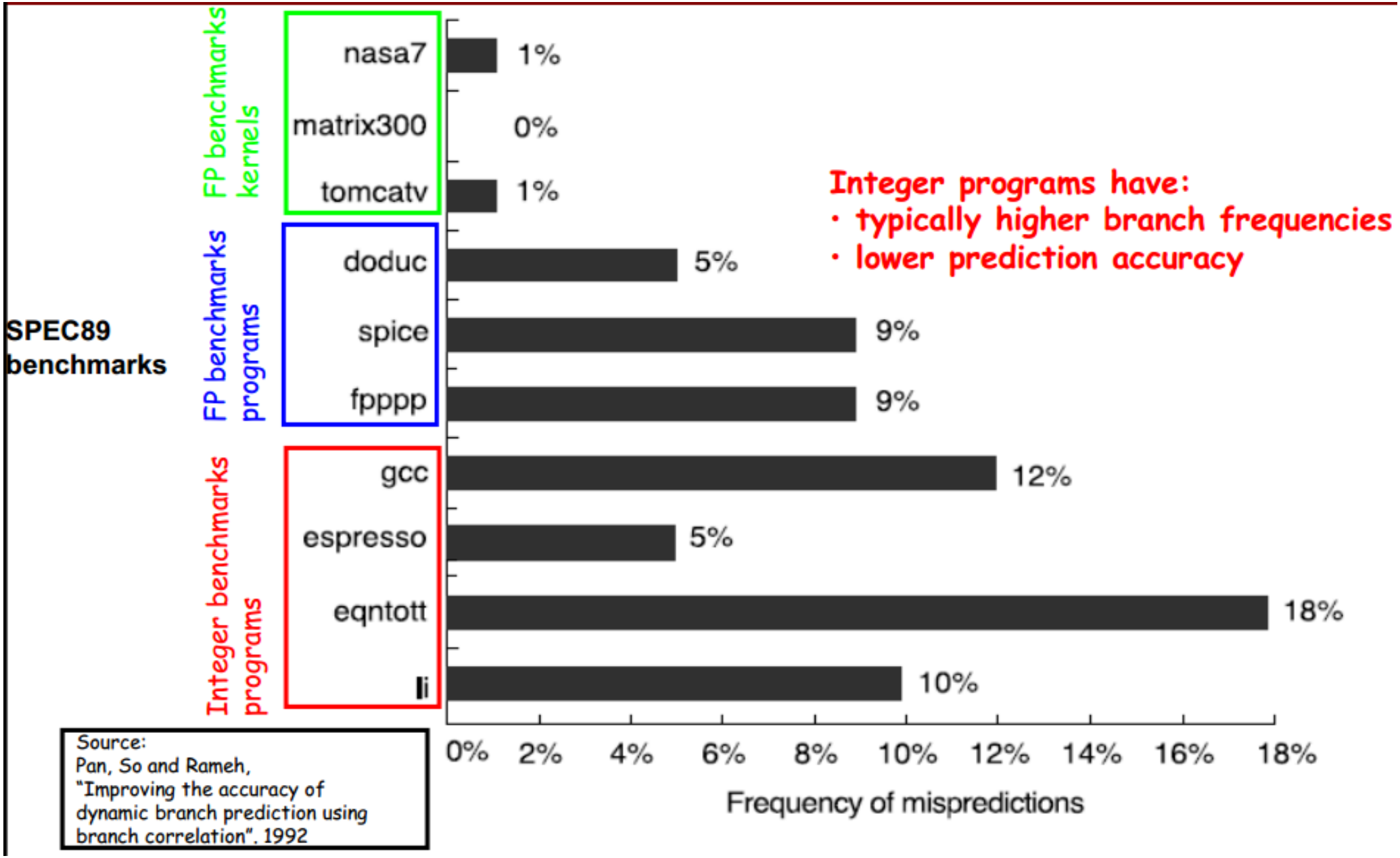
Iterationss & steps	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4	5	6	7	8	9	10
Predicted behavior	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
Actual behavior	T	T	T	T	T	T	T	T	T	T	NT	T	T	T	T	T	T	T	T	T	T	NT

- Branch taken 90% of the times, and branch prediction accuracy is now 90%
- The 2-bit predictor mispredicts at the 10th step of the 1st iteration, but *doesn't change its mind*. It just moves to "weak-predict-taken" for the 1st step of the following iteration

Branch History Table for 4K-Entry 2-Bit Predictor: 4K Entry □ 2 bits/Entry = 8K bit



Measure: Prediction Accuracy of 4K-entry 2-bit-prediction buffer with SPEC89



Example

A snapshot of the taken/not-taken behavior of a branch is:

... T T T T T T T T **N N T T N N T N N T**

If the branch predictor used is a 2-bit saturating counter, how many of the last ten branches are predicted correctly?

Answer：

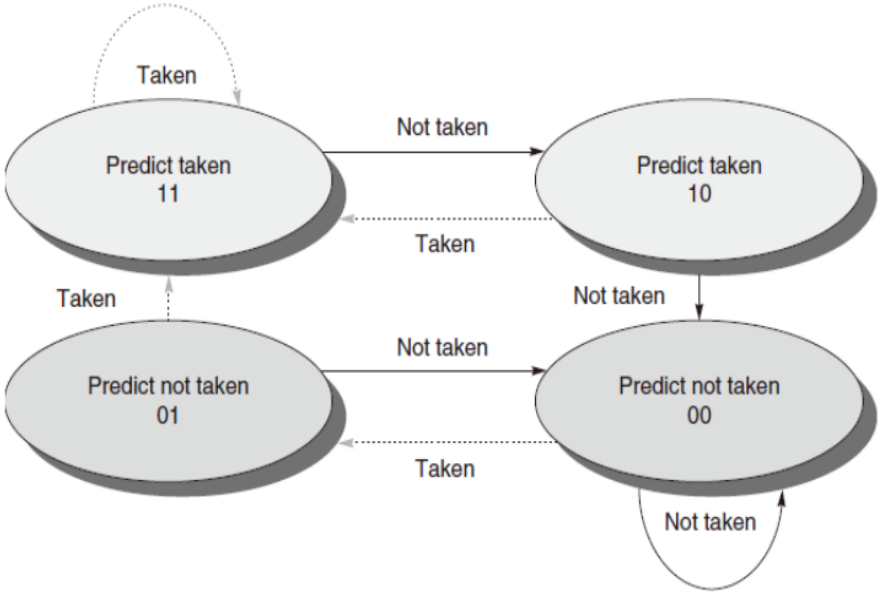
According to the branch predictor in textbook, the prediction for the last ten branches are:

ST, WT, SN, WN, ST, WT, SN, WN, SN, SN

According to the 2-bit saturating counter:

ST, WT, WN, WT, ST, WT, WN, WT, WN, SN

No matter which one you use, the answer is 2 branches are predicted correctly.



Predictor state	Prediction	Action	Correct?
11	T	N	N
10	T	N	N
00	N	T	N
01	N	N	Y
00	N	T	N
01	N	T	N
11	T	N	N
10	T	N	N
00	N	T	N
01	N	N	Y

Correct prediction: twice.

Correlating Branch Predictors

```
if (aa==2)
        aa=0;
if (bb==2)
        bb=0;
if (aa!=bb) {
```

The limitation of the basic 2-bit predictor:

The 2-bit predictor schemes use only the recent behavior of **a single branch** to predict the future behavior of that branch. Increasing to 3-bit or more does not help much!

How about looking at the recent behavior of other branches?

Look the code on the top-right corner: if the first two branches are taken, the 3rd is never taken.

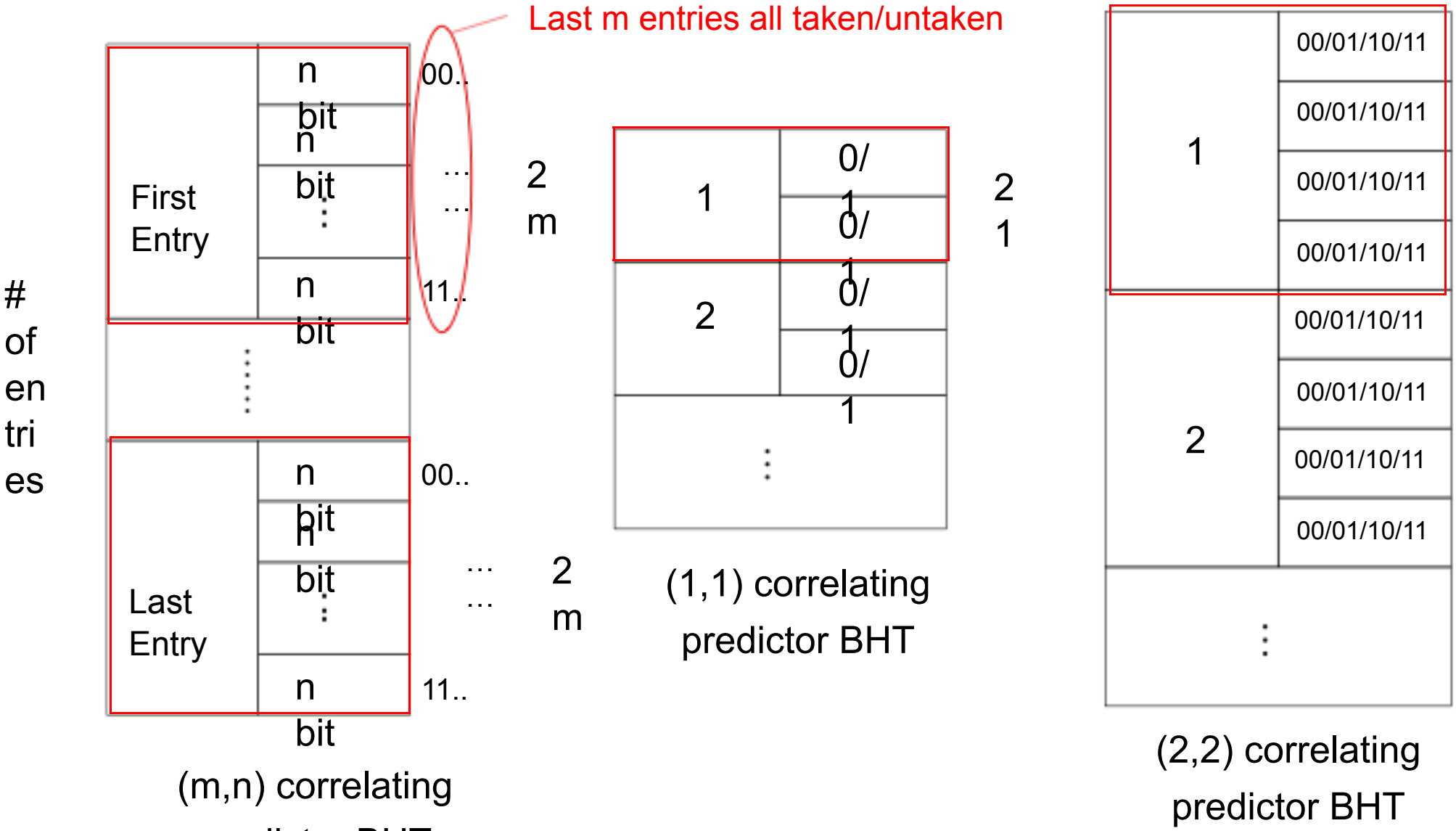
Correlating predictor / 2-level predictor:

Adds information of the most recent branches to decide how to predict a given branch. → 前 m 个 branch 相关

An (m,n) 2-level predictor uses the behavior of the last m branches to choose from 2m branch predictors, each of which is an n-bit predictor for a single branch. → n 个 bit 表示信息

The size of BHT = # of Entries □ # of bits / Entry = # of Entries ~~□ 2m □~~ × 2^m × n n

Correlating Branch Predictors



Adding "Global" Information to Branch Prediction: Correlating (or Two-Level) Branch Predictors

```
if (a == 2)
  a = 0;
…
if (b == 2)
  b = 0;
…
if (a != b) {
  …
```

```
      DSUBUI R3, R1, #2
b1 →  BNEZ R3, L1
      DADD R1, R0, R0
L1:   DSUBUI R3, R2, #2
b2 →  BNEZ R3, L2
      DADD R2, R0, R0
L2:   DSUBU R3, R1, R2
b3 →  BEQZ R3, L3
```

- Prediction accuracy can be improved by accounting for branch spatial correlations and looking at the behavior of other branches
  - e.g. in the example above, if the first two branches (b1 and b2) are not taken then the third (b3) will be taken

A Simple Example: Set-Up

```
if (d == 0)
  d = 1;
if (d == 1) {
  …
```

```
b1 → BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1,  #1
b2 → BNEZ R3, L2
     …
L2:  …
```

init d	d==0?	b1	d before b2	d==1?	b2
0	yes	not taken	1	yes	not taken
1	no	taken	1	yes	not taken
x ≠{0,1}	no	taken	x ≠ {0,1}	no	taken

Correlation: if b1 is not taken then d is set to 1 and b2 is also not taken

A Simple Example: 1-Bit Predictor

```
if (d == 0)
  d = 1;
if (d == 1) {
  …
```

```
b1 → BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1,  #1
b2 → BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
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

A 1-bit predictor that is initialized to not-taken mispredicts all branches

A Simple Example: 1-Bit Predictor with 1-Bit Correlation {i.e., a (1,1) Predictor}

```
if (d == 0)
  d = 1;
if (d == 1) {
  …
```

```
b1 → BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #1
b2 → BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0

X/Y: use X if last branch was not taken, use Y if last branch was taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
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

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 0

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

b1

b2

Assumption: d alternates between 2 and 0

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
2	NT/NT	T		NT/NT	T	
0		NT			NT	
2		T			T	
0		NT			NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 1

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0

X/Y: use X if last branch was not taken, use Y if last branch was taken

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
2	NT/NT	T	T/NT	NT/**NT**	T	
0	T/NT	NT			NT	
2		T			T	
0		NT			NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 2

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0

X/Y: use X if last branch was not taken, use Y if last branch was taken

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
2	NT/NT	T	T/NT	NT/NT	T	NT/T
0	T/NT	NT		NT/T	NT	
2		T			T	
0		NT			NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 3

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0

X/Y: use X if last branch was not taken, use Y if last branch was taken

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
2	NT/NT	T	T/NT	NT/NT	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	
2	T/NT	T			T	
0		NT			NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 4

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0

X/Y: use X if last branch was not taken, use Y if last branch was taken

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
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		NT/T	T	
0		NT			NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 5

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0
X/Y: use X if last branch was not taken,
use Y if last branch was taken

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
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	
0	T/NT	NT			NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 6

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0
X/Y: use X if last branch was not taken,
use Y if last branch was taken

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
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		NT/T	NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 7

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0

X/Y: use X if last branch was not taken, use Y if last branch was taken

Initial Status

b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
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	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

A Simple Example of (1,1) Predictor: Simulation - Cycle 8

```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

Assumption: d alternates between 2 and 0
X/Y: use X if last branch was not taken,
use Y if last branch was taken

Initial Status

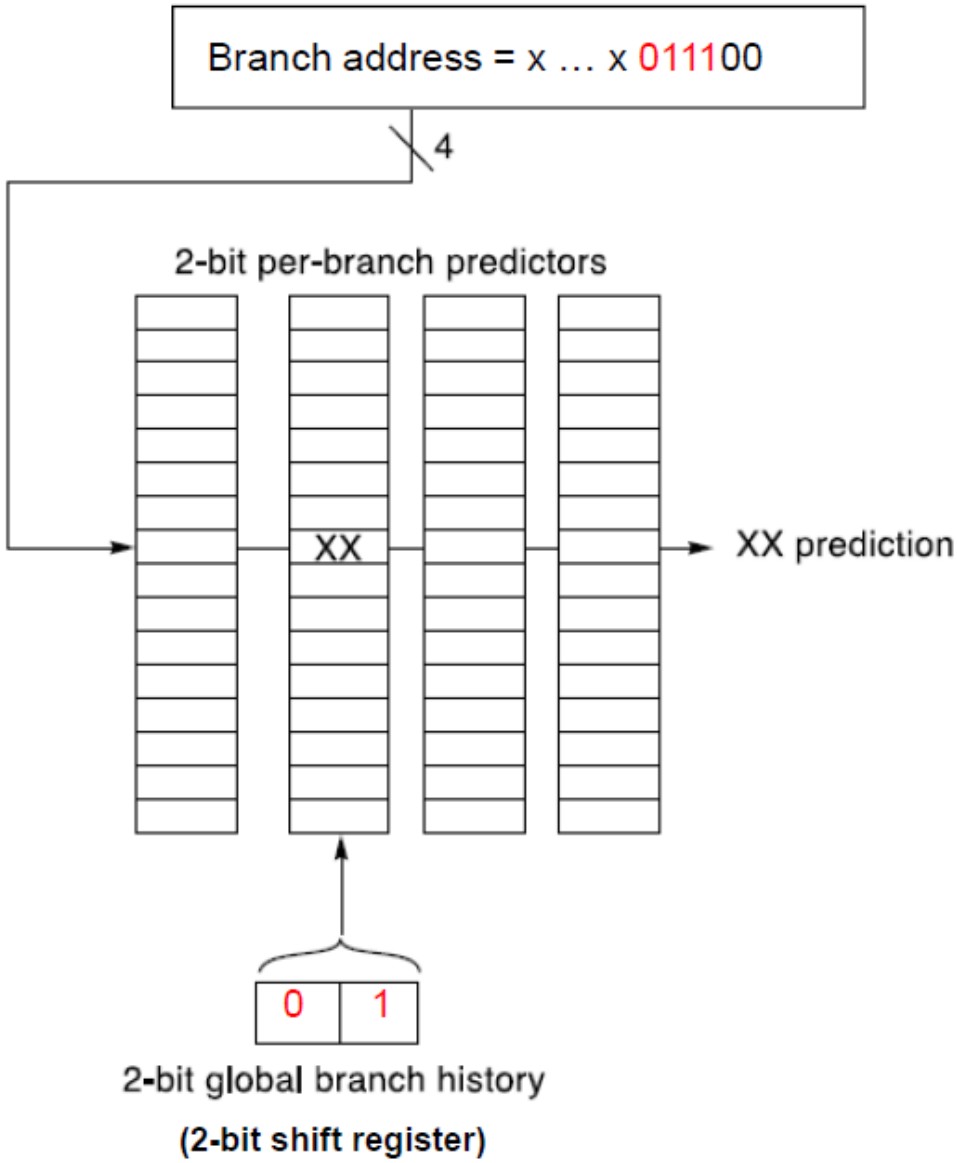
b1	NT 0	b1's last branch untaken
	NT 1	b1's last branch taken
b2	NT 0	b2's last branch untaken
	NT 1	b2's last branch taken

d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
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

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

Example:
A 64-Entry (2,2) Branch Prediction Buffer

- Global history of the most recent m=2 branches is recorded in an 2-bit shift register where each bit records whether the branch was taken or not taken
- A concatenation of the low-order bits of the branch instruction address and the 2-bit global history is used to index the buffer and get the (n=2)-bit predictor
- Concept can be generalized to (m,n) predictor

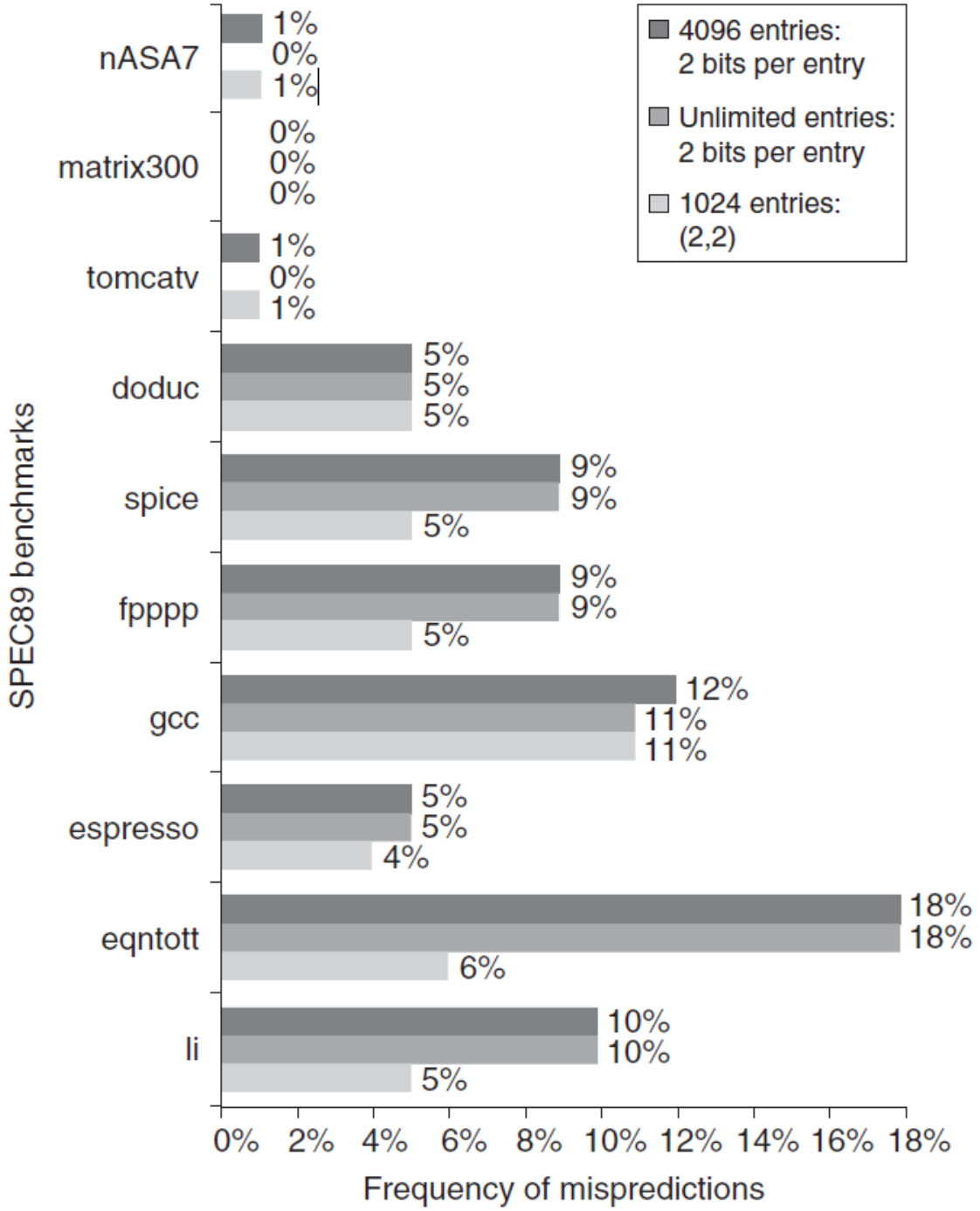


Compare 2-bit vs Correlating

Three are in comparison:

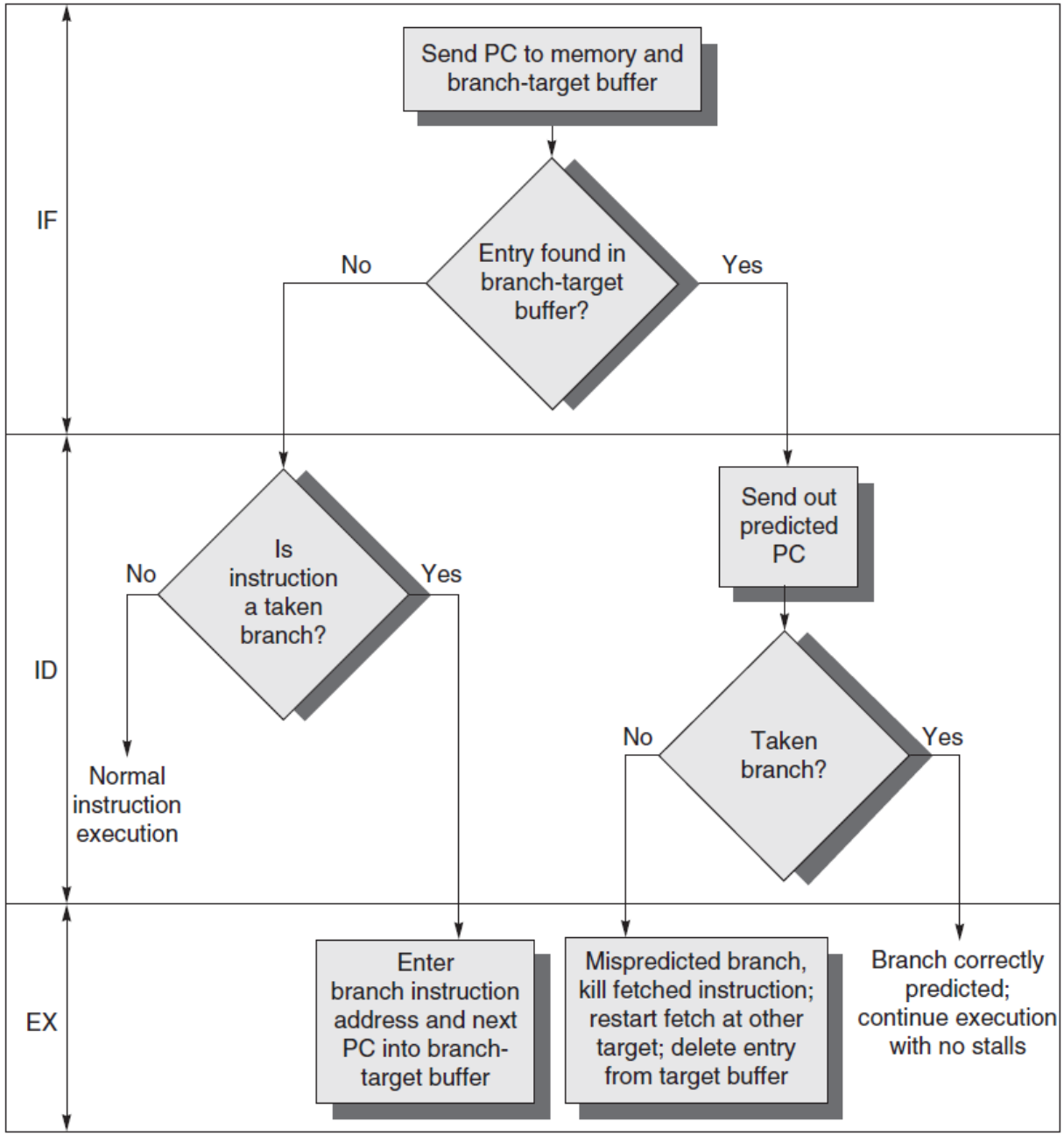
- 2-bit Predictor, (0,2)
- (2,2) Correlating
- 2-bit Predictor with unlimited entries.

The first two have the same total number of bits in their BHT.



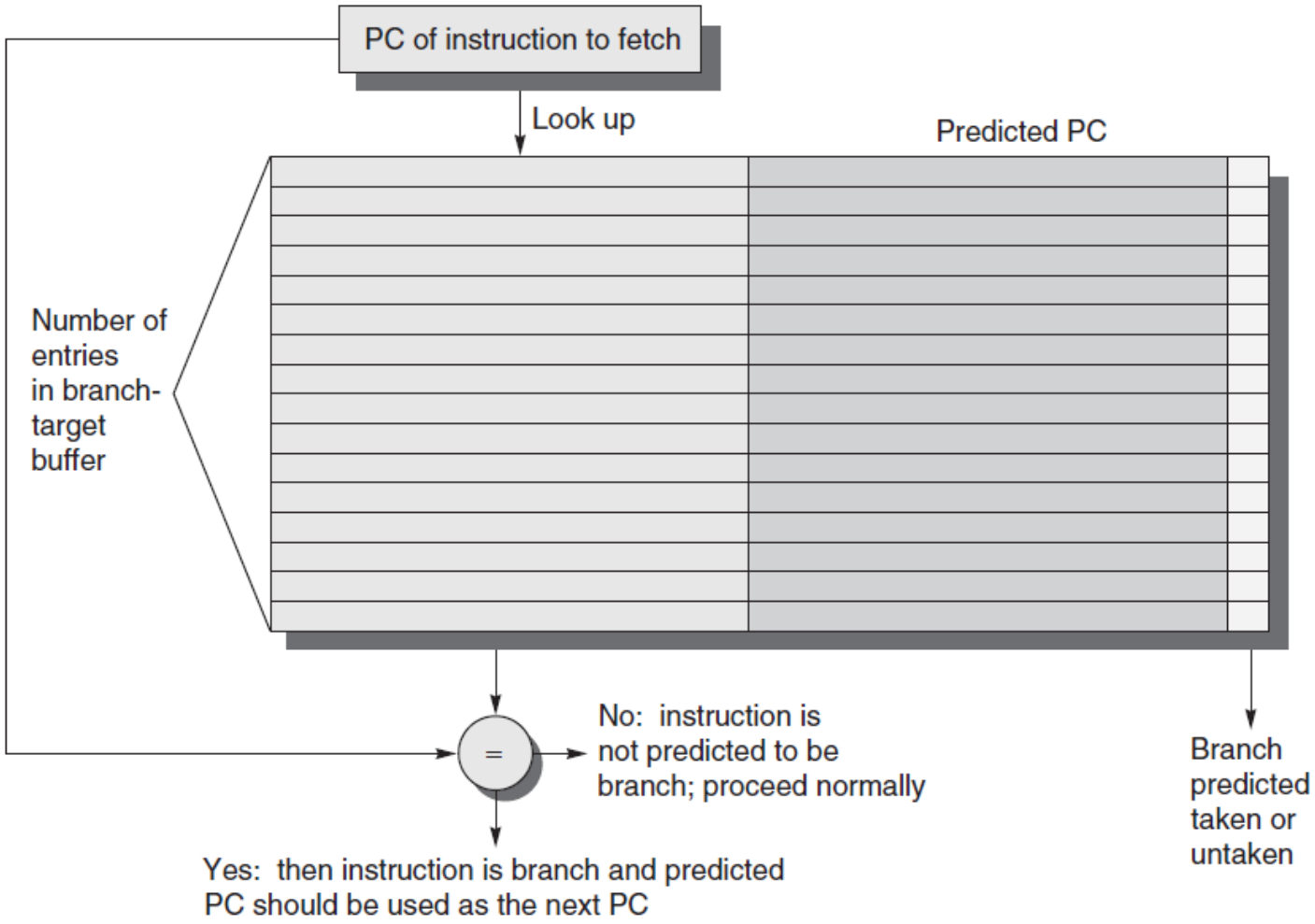
Predicting the Instruction Target Address: Branch-Target Buffer (or Branch-Target Cache)

- **Goal**: learn the **predicted instruction address at the** end of IF stage
  - one cycle earlier
    - at best, branch-prediction buffers know the next predicted instruction at the end of ID
- **No branch delay**
  - if entry found in buffer and prediction is correct
  - if entry not found and branch is not taken
- **Two cycle penalty**
  - if prediction is incorrect
  - if entry is not found and branch is taken



Predicting the Instruction Target Address: Branch-Target Buffer (or Branch-Target Cache)

- Only necessary to store the Predicted taken branches since an untaken branch follows the same strategy (to fetch the fall-through instruction) as a non-branch instruction
- But using a 2- bit predictor requires to store information also for untaken branches



Branch Target Buffer: Penalty Table

Instruction in buffer	Prediction	Actual Branch	Penalty Clock Cycles
yes	taken	taken	0
yes	taken	not taken	2
no		taken	2
no		not taken	0

- Assuming
  - –85% prediction accuracy, 90% buffer hit rate, 60% branches taken

- P(branch not in buffer, but taken) = 0.1 x 0.6 = 0.06
- P(branch in buffer but not taken) = 0.9 x 0.15 = 0.135
- Total branch penalty = (0.135 + 0.06) x 2 = 0.39
  - – the penalty for delayed-branch was about 0.5 clock cycles
  - – penalty is lower with better predictors (and bigger branch delays)

Example

Assume a machine that has a branch-target buffer with 8 entries. A branch in this machine has a penalty of 2 clock cycles if the branch is taken and the target instruction is not in the branch-target buffer, or if the branch is predicted as taken, the instruction is in the branch-target buffer, but the branch is actually not taken. In all other situations the branch penalty is zero. What is the total branch penalty in this machine, measured in clock cycles, if

- the branch prediction accuracy is 90%;
- 80% of the time the target instruction is in the buffer (80% hit rate in the buffer);
- 60% of the branches are actually taken.

Answer：

Probability of branch taken but not found in the buffer:

Percentage of taken branches * buffer miss rate = 60% * 20% = 0.12

Probability of branch found in buffer but predicted wrong:

Buffer hit rate * prediction miss rate = 80% * 10% = 0.08

Average branch penalty = (0.12 + 0.08) * 2 = 0.4 clock cycles