

Branch Prediction Techniques

M. Sonza Reorda

**Politecnico di Torino
Dipartimento di Automatica e Informatica**

1

INTRODUCTION

Branches can potentially impact in a very serious way the pipeline performance.

It is possible to reduce the performance losses by predicting how branches will behave.

2

Branch prediction schemes

They aim at correctly forecasting branches, thus reducing the chance that control dependences cause stalls.

They can be categorized in two groups:

- *static techniques*: they are handled by the compiler resorting to a preliminary analysis of the code
- *dynamic techniques*: they are implemented by the hardware based on the behavior of the code.

3

Static branch prediction

It can be useful when combined with other static techniques, such as

- Delayed branches
- Rescheduling to avoid data hazards.

4

Static branch prediction

The compiler may predict branch behavior in different ways:

- Always predicting branches as taken
- Predicting branch behavior depending on branch direction
- Predicting on the basis of profile information coming from earlier runs.

5

Predicting branches as taken

On the SPEC programs this gives:

- 34% average misprediction rate
- highly variable rate (from 9% to 59%).

Other techniques may behave better in the average, but still with very high variations from program to program.

6

Predicting branch behavior depending on branch direction

This technique is based on the observation that

- Forward branches are more often untaken
- Backward branches are more often taken.

This behavior is mainly due to loop constructs.

7

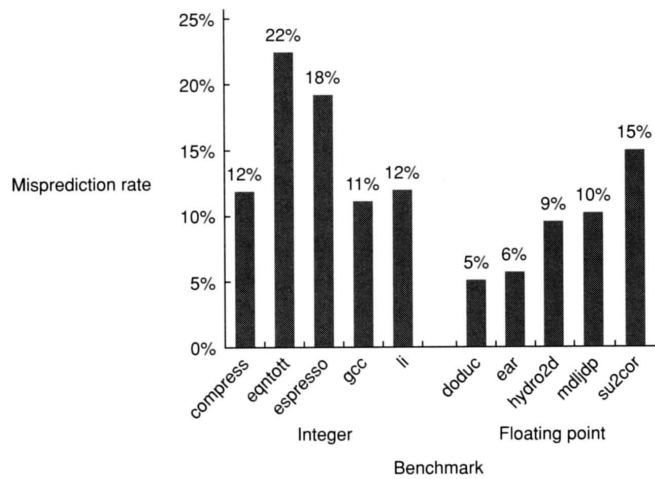
Predicting branches based on profile information

This technique is based on

- Identifying a typical sequence of input stimula for the program
- Running the program a limited number of times using these stimula
- Gathering statistics on the behavior of branches
- Using these statistics as predictions.

8

Profile-based prediction effectiveness



9

Dynamic branch prediction

It can be based on 3 techniques:

- Branch history table
 - One- and two-bit prediction schemes
 - Two-level prediction schemes
- Branch-target buffer.

10

BRANCH HISTORY TABLE

It is the simplest method for dynamic branch prediction.

The *Branch History Table* (BHT) is a small memory:

- indexed by the lower portion of the address of the branch instruction
- containing one bit recording whether the branch has been taken or not the last time it has been executed.

11

Algorithm

Each time a branch is decoded

- An access is made in the BHT using the lowest portion of its address
- The prediction stored in the table is used.

When the branch result is known, the BHT is possibly updated.

12

Effectiveness

The effectiveness of the method depends on:

- the chance that the memory entry relates to the branch of interest
- the accuracy of the prediction.

13

Example

Consider a loop branch which is taken nine times in a row, then not taken once. Assume that the entry for the branch is not shared with other branches.

The steady-state prediction behavior will mispredict on the first and last loop iterations.

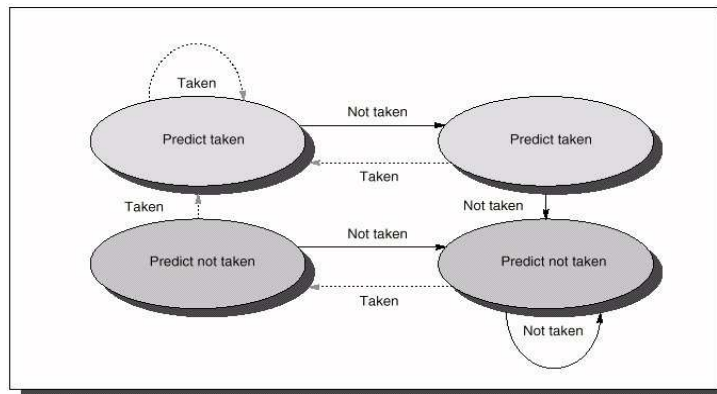
The prediction accuracy is thus 80%, lower than simply assuming that the branch is taken (90%).

14

Two-bit Prediction Schemes

They provide higher prediction capabilities.

For every branch, two bits are maintained, and the prediction is changed only after missing twice.



15

Effectiveness

The described technique works as follows:

- When the decode stage identifies a branch instruction, activates the prediction logic
- The new PC used by the IF stage is determined according to the prediction
- When the result of condition evaluation is available, the prediction is checked for correctness.

In the MIPS processor, condition evaluation is performed while branch instructions are identified. Therefore, the described technique does not give any advantage.

16

n-bit Prediction Scheme

It is the general case of the previous one.

It is based on an n-bit saturating counter associated to every branch instruction.

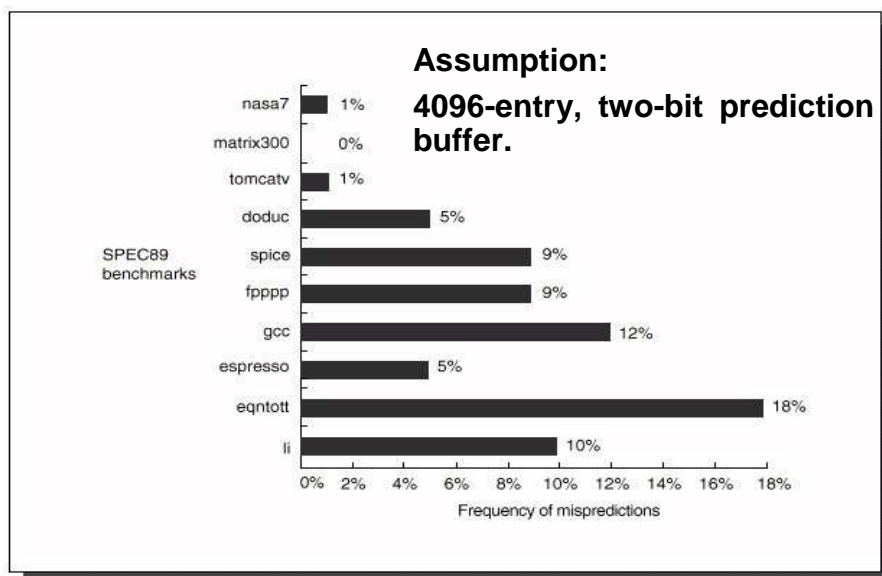
The counter is incremented each time the branch is taken, decremented each time it is not.

When the counter value is greater than one half of its maximum value, the branch is predicted as taken, when it is lower it is predicted as not taken.

Experiments have shown that there is little advantage in using $n > 2$.

17

Prediction Accuracy for SPEC89



18

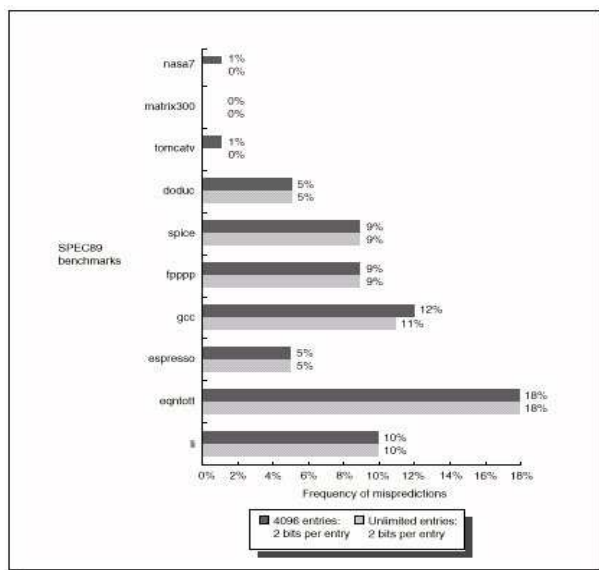
Performance impact of branches

It depends on

- prediction accuracy
- branch cost (penalty for misprediction)
- branch frequency (lower for FP programs).

19

Dependence on Buffer Size



20

Experimental Evidence

Prediction accuracy does not grow significantly by increasing

- the buffer size
- the number of bits per predictor.

21

CORRELATING PREDICTORS

This approach (also called *two-level predictors*) is based on exploiting the dependencies between the results of branches.

Example

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

The behavior of the last branch is strongly dependent on the result of the previous ones.

22

(1,1) predictor

Each branch is associated with two bits:

- one reporting the prediction in the case the previous branch was taken
- one with the prediction in the case the previous branch was not taken.

Warning: the two branches the prediction is based on can be different.

23

(m,n) predictors

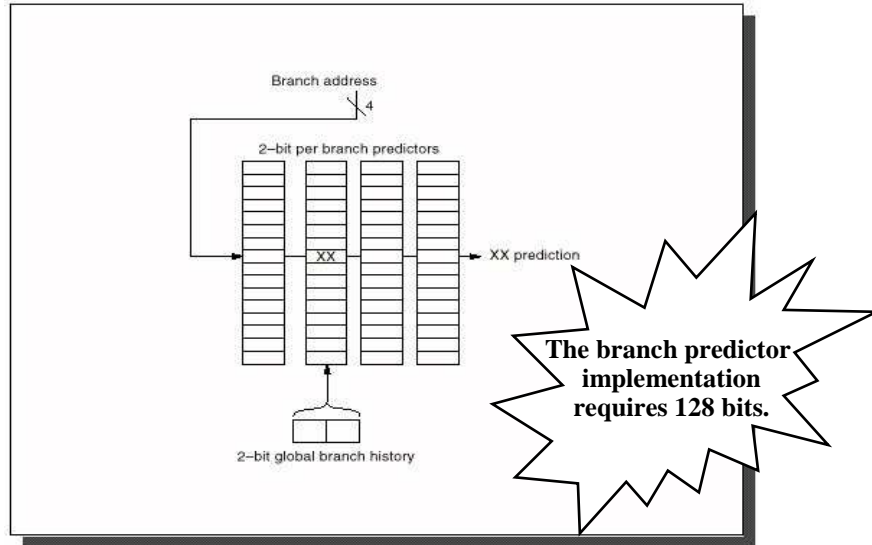
They use the behavior of the last m branches to choose from 2^m branch predictors, each of which is a n -bit predictor.

The hardware required for implementing this scheme is very simple:

- The history of the most recent m branches is recorded in an m -bit shift register, where each bit records whether the branch was taken or not.
- The branch-prediction buffer is indexed using a concatenation of the low-order bits from the branch address with the m low-order history bits.

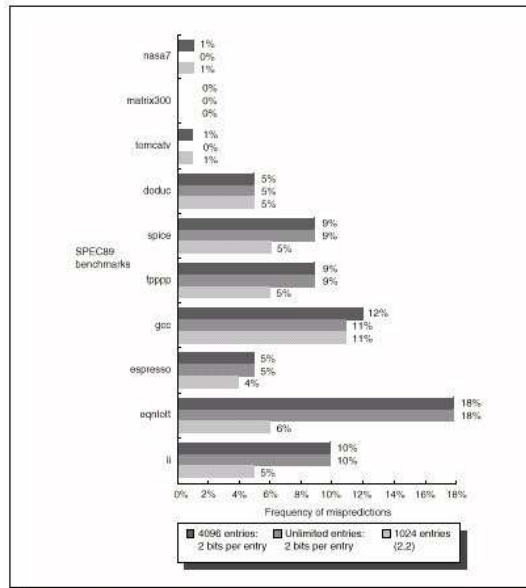
24

(2,2) predictor implementation



25

Performance comparison



26

BRANCH-TARGET BUFFER

Reducing the negative effects of control dependencies requires knowing as soon as possible

- whether the branch has to be taken or not
- the new value of the PC (if the branch is assumed to be taken).

The later issue is faced by introducing a *branch-target buffer* (or *cache*).

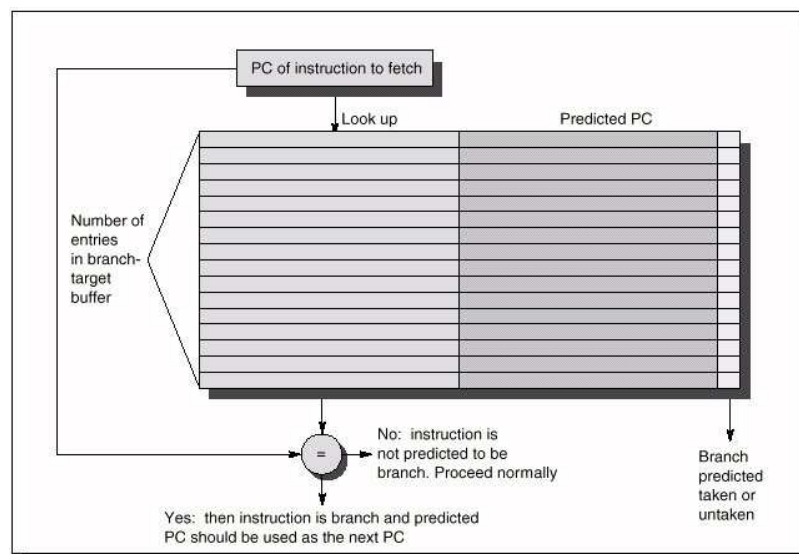
The branch-target buffer contains

- the addresses of the last executed branches
- the values to be loaded in the PC.

Using the branch-target buffer, the PC is loaded with the new value at the end of the IF stage, i.e., even before the branch instruction is decoded.

27

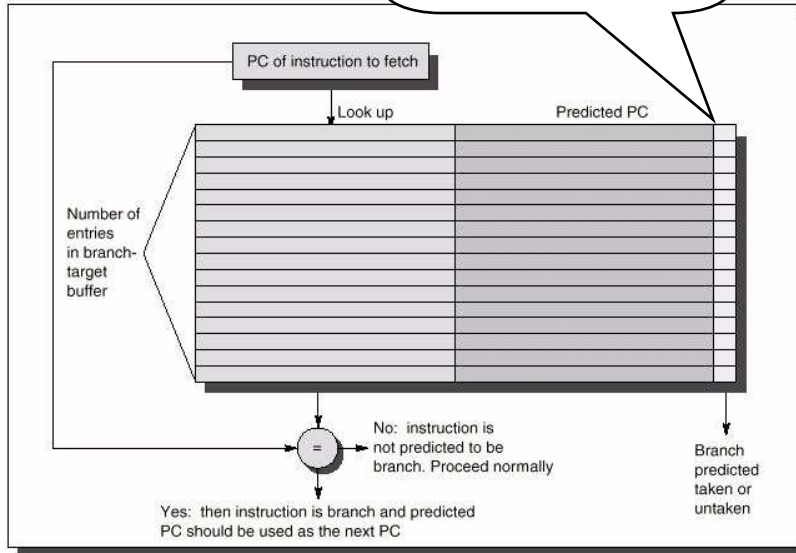
Branch-target Buffer: Architecture



28

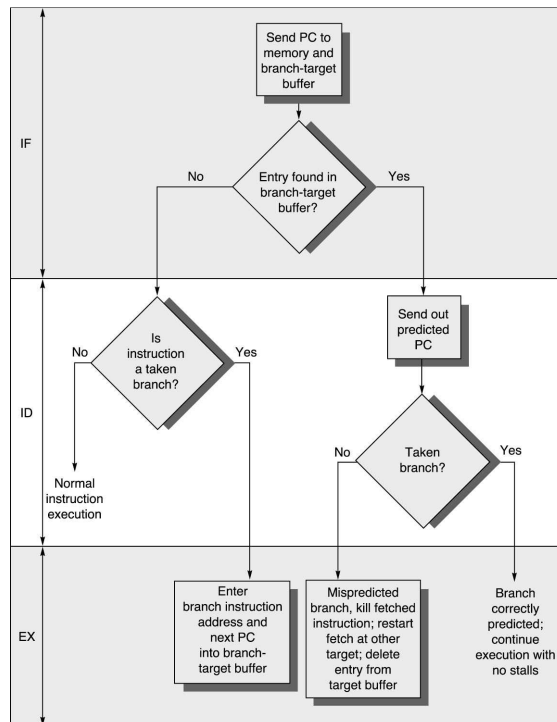
Branch-target Buffer Architecture

This field is not strictly necessary.



29

Branch-target Buffer: Behavior



30

Branch-target Buffer: Advanced Issues

If a two-bit prediction strategy is adopted, it is possible to combine a branch-target buffer with a branch prediction buffer.

There are ways for extending the branch-target buffer technique to indirect target addresses.

31

Branch-target buffer: Performance Effects

Let assume the following penalty parameters

Instruction in buffer	Prediction	Actual branch	Penalty cycles
Yes	taken	taken	0
Yes	taken	not taken	2
No		taken	2
No		not taken	0

Let also assume that

- the prediction accuracy is 90%
- the hit rate in the buffer is 90%.

Which is the total branch penalty?

32

Solution

$$\begin{aligned} \text{Branch Penalty} = & (\text{percent buffer hit rate} \times \\ & \text{percent incorrect predictions} \times 2) + \\ & ((1 - \text{percent buffer hit rate}) \times \\ & \text{taken branches} \times 2) = \\ & (90\% \times 10\% \times 2) + (10\% \times 60\% \times 2) = \\ & 0.18 + 0.12 = 0.30 \end{aligned}$$

This figure should be compared with the 0.50 clock cycles per branch penalty existing with delayed branches.

33

Storing instructions instead of addresses

In some cases the branch-target buffer contains instructions instead of addresses.

This allows:

- to avoid fetching an instruction
- to support *branch folding*: the branch instruction is eliminated, since its effects (changing the PC value) are obtained by simply accessing to the buffer.

34