Lecture 3: Branch prediction

Hossam A. H. Fahmy

Cairo University, Faculty of Engineering

Overview

1. Control Hazards
2. Speculation
   - Static prediction
   - Dynamic prediction
   - Two bit predictors
   - Branch correlations
3. Conclusions

Branches

There are two important questions.

- When is the target address ready?
- When is the condition resolved?

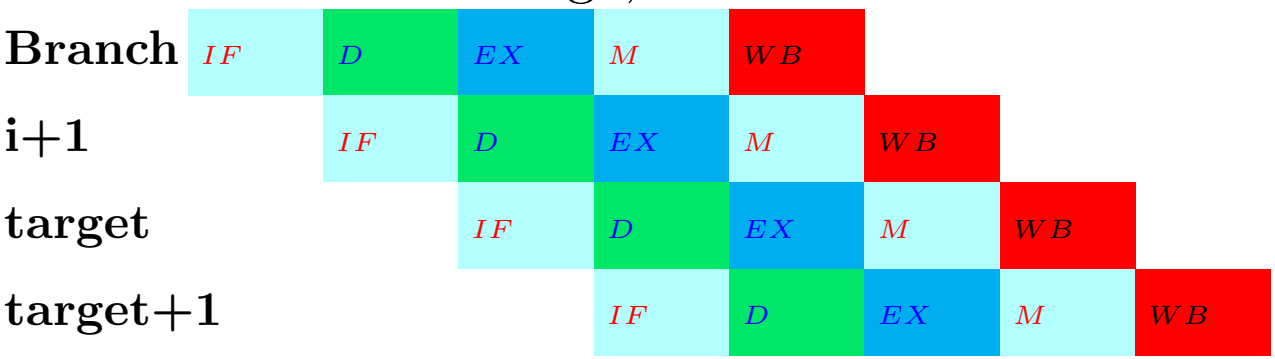
In a simple RISC processor with simplified branch instructions, we might add more hardware in the decode stage to

- calculate the target address and
- resolve the simple conditions.

Downside: complex tests must be done by separate instructions and the decode stage is even more complicated now.

Delayed branches

Assuming a fast branch, with all the information known by the end of the decode stage, what shall I do with instruction i+1?



- Flush it and lose one cycle.
- Assume that the effect of the branch is delayed by one cycle and bring something from before the branch to put it in this slot. (Maybe even from after it as long as it is not "harmful".)

⇒ Delayed branches expose the internal organization to the compiler writer and complicates interrupt handling. Moreover, most processors have complicated branches where the result is not known till a few cycles pass.

Speculation

We have already seen a few 'solutions' to branches:

- stall,
- fast branches evaluated completely at the decode stage, and
- delayed branches.

Can we do better by *predicting* the decision and executing *speculatively*?

The speculation game

1. Guess the branch target.
2. Execute the branch to verify the guess.
3. Meanwhile, start execution at the guessed position.

We should attempt to minimize the penalty if our guess is right to almost zero. If our guess is wrong the penalty might be higher.

$\Rightarrow$ How often is our guess correct?

$\Rightarrow$ How can we improve this probability?

$\Rightarrow$ What are the penalties?

Branch prediction

It is one of the heavily researched areas in computer architecture during the 1990s.

Fixed: As an example, a processor may always fetch in-line on true conditional branches.

Static: The strategy varies by opcode type but is predetermined.

Dynamic: The strategy varies according to the program behavior and depends on the history of this branch. $\Rightarrow$ Use up-down saturating counters.

Perfect prediction simply converts the delay for conditional branch into that for unconditional branch (branch taken). *The important goal is the minimization of the branch delay not just a higher prediction accuracy.*

Static options

The prediction to continue in-line is quite easy since we already know the target (it is at $PC + 4$). However, this is not efficient.

Most branches (especially backward) are taken.

- Generate the profile of your target applications.
- Find out the most probable decision for each type of branch.
- Implement the hardware to predict statically based on this information.

Dynamic behavior

- Was this branch taken or not taken in the past iterations?
- How many iterations shall we consider?

The simplest case is to look for the last time only in a *branch history table (BHT)*.

⇒ Use the least significant bits of the PC to index a small table.

⇒ Each entry in the table is one bit indicating if the last time was taken or not. Use this bit as your prediction.

⇒ In case of a misprediction, complement this bit.

Problems of the BHT

The BHT is simple to implement but

- multiple PCs may alias to the same location and
- we may have many mispredictions.

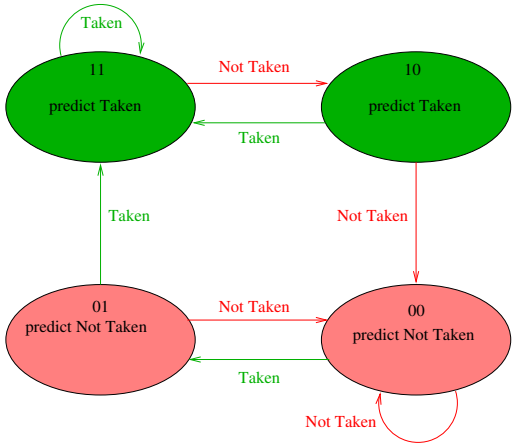
Let us look at an inner loop with four iterations.

Prediction	N	T	T	T	N	T	T	T	N	T	T	T
branch outcome	T	T	T	N	T	T	T	N	T	T	T	N
Misprediction?	+			+	+			+	+			+

We get two misprediction in four decisions.

Improvement: 2-bit prediction

What if we look at the last two iterations? A prediction is not changed untill it misses twice.



The inner loop with four iterations.

Prediction	N	N	T	T	T	T	T	T	T	T	T	T
branch outcome	T	T	T	N	T	T	T	N	T	T	T	N
Misprediction?	+	+		+				+				+

We get only one misprediction in four decisions.

Another 2-bit idea

We can use a saturating up-down two bit counter.

strong Not taken	weak not taken	weak taken	strong Taken
00	01	10	11
N	n	t	T

The inner loop with four iterations once more.

Prediction	n	t	T	T	t	T	T	T	t	T	T	T
branch outcome	T	T	T	N	T	T	T	N	T	T	T	N
Misprediction?	+			+				+				+

We get only one misprediction in four decisions.
This can be easily extended to more than two bits although 2-bit predictors are good enough for many systems.

Branch prediction buffer

- This is a table accessed during the fetch cycle by the least significant bits of the PC.
- Each entry may be a 2-bit predictor.
- If we decode the instruction to be a branch, we have the prediction for it.

This leads to a predicition accuracy in the high 80% to over 99% in some applications.

Correlating predictors

```
1 if (aa==2)
2         aa = 0;
3 if (bb==2)
4         bb = 0;
5 if (aa!=bb){ .... }
```

- The decision on the third branch depends on the previous two.
- We can correlate the branches!

Global history

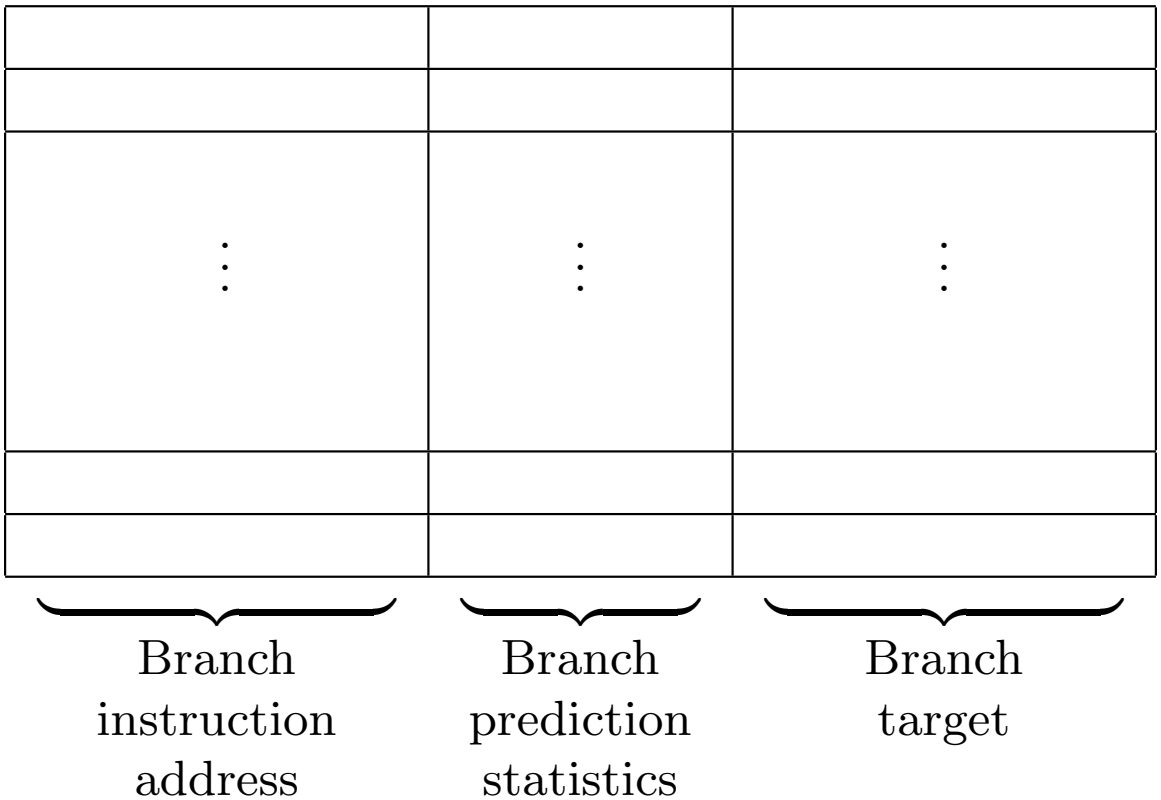
- A *branch history shift register* keeps the history of the previous few branches. One bit for each branch. For example, for a history of two branches we have two bits and four possible cases: 00, 01, 10, and 11.
- We have a table (of the branch prediction buffer) for each case and decide the prediction on the least significant bits of the PC *and* the global history.

Hybrid schemes

Different predictors work best for different branches.

$\Rightarrow$ Let us combine them and choose the best.

Branch Target Buffer



If the IF "hits" in the BTB, the target instruction that was previously stored in the BTB is now fetched and forwarded to the processor at its regularly scheduled time.

Conclusions

- There are many ways to minimize the effect of control hazards.
- We can achieve very high prediction ratios.
- We must also minimize the penalty of the branch.