Lecture 8: Branch Prediction, Dynamic ILP

- Topics: branch prediction, out-of-order processors (Sections 2.3-2.6)
Control Hazards

- In the 5-stage in-order processor: assume always taken or assume always not taken; if the branch goes the other way, squash mis-fetched instructions (momentarily, forget about branch delay slots)

- Modern in-order and out-of-order processors: dynamic branch prediction; instead of a default not-taken assumption, either predict not-taken, or predict taken-to-X, or predict taken-to-Y

- Branch predictor: a cache of recent branch outcomes
In the 5-stage pipeline, a branch completes in two cycles → If the branch went the wrong way, one incorrect instr is fetched → One stall cycle per incorrect branch
Pipeline with Branch Predictor

In the 5-stage pipeline, a branch completes in two cycles →
If the branch went the wrong way, one incorrect instr is fetched →
One stall cycle per incorrect branch
Branch Mispredict Penalty

- Assume: no data or structural hazards; only control hazards; every 5\textsuperscript{th} instruction is a branch; branch predictor accuracy is 90%

- Slowdown = \frac{1}{1 + \text{stalls per instruction}}

- Stalls per instruction = \% branches \times \%mispreds \times \text{penalty} = 20\% \times 10\% \times 1 = 0.02

- Slowdown = \frac{1}{1.02} ; \text{if penalty} = 20, \text{slowdown} = \frac{1}{1.4}
1-Bit Bimodal Prediction

• For each branch, keep track of what happened last time and use that outcome as the prediction

• What are prediction accuracies for branches 1 and 2 below:

```java
while (1) {
    for (i=0;i<10;i++) {                     branch-1
        ...
    }
    for (j=0;j<20;j++) {                     branch-2
        ...
    }
}
```
2-Bit Bimodal Prediction

- For each branch, maintain a 2-bit saturating counter:
  - if the branch is taken: counter = min(3,counter+1)
  - if the branch is not taken: counter = max(0,counter-1)

- If (counter >= 2), predict taken, else predict not taken

- Advantage: a few atypical branches will not influence the prediction (a better measure of “the common case”)

- Especially useful when multiple branches share the same counter (some bits of the branch PC are used to index into the branch predictor)

- Can be easily extended to N-bits (in most processors, N≠2)
Bimodal 1-Bit Predictor

Branch PC

10 bits

Table of 1K entries

Each entry is a bit

The table keeps track of what the branch did last time
Bimodal 2-Bit Predictor

Branch PC

10 bits

Table of 1K entries

Each entry is a 2-bit sat. counter

The table keeps track of the common-case outcome for the branch
Correlating Predictors

- Basic branch prediction: maintain a 2-bit saturating counter for each entry (or use 10 branch PC bits to index into one of 1024 counters) – captures the recent “common case” for each branch

- Can we take advantage of additional information?
  - If a branch recently went 01111, expect 0; if it recently went 11101, expect 1; can we have a separate counter for each case?
  - If the previous branches went 01, expect 0; if the previous branches went 11, expect 1; can we have a separate counter for each case?

Hence, build correlating predictors
Global Predictor

The table keeps track of the common-case outcome for the branch/history combo.
Local Predictor

Branch PC

Use 6 bits of branch PC to index into local history table

Table of 64 entries of 14-bit histories for a single branch

10110111011001

14-bit history indexes into next level

Also a two-level predictor that only uses local histories at the first level

Table of 16K entries of 2-bit saturating counters
Local Predictor

The table keeps track of the common-case outcome for the branch/local-history combo
Local/Global Predictors

• Instead of maintaining a counter for each branch to capture the common case,

→ Maintain a counter for each branch and surrounding pattern
→ If the surrounding pattern belongs to the branch being predicted, the predictor is referred to as a local predictor
→ If the surrounding pattern includes neighboring branches, the predictor is referred to as a global predictor
Tournament Predictors

- A local predictor might work well for some branches or programs, while a global predictor might work well for others.

- Provide one of each and maintain another predictor to identify which predictor is best for each branch.

\[\text{Table of 2-bit saturating counters}\]

\[\text{Local Predictor}\]

\[\text{Global Predictor}\]

\[\text{Tournament Predictor}\]

\[\text{MUX}\]

\[\text{Branch PC}\]

\[\text{Alpha 21264:}\]
\[1K \text{ entries in level-1}\]
\[1K \text{ entries in level-2}\]

\[4K \text{ entries}\]

\[12\text{-bit global history}\]

\[4K \text{ entries}\]

\[\text{Total capacity: ?}\]
Branch Target Prediction

• In addition to predicting the branch direction, we must also predict the branch target address

• Branch PC indexes into a predictor table; indirect branches might be problematic

• Most common indirect branch: return from a procedure – can be easily handled with a stack of return addresses
An Out-of-Order Processor Implementation

- **Branch prediction and instr fetch**
- **Instr Fetch Queue**
  - \( R1 \leftarrow R1+R2 \)
  - \( R2 \leftarrow R1+R3 \)
  - \( \text{BEQZ } R2 \)
  - \( R3 \leftarrow R1+R2 \)
  - \( R1 \leftarrow R3+R2 \)

- **Decode & Rename**

- **Reorder Buffer (ROB)**
  - \( \text{Instr 1} \rightarrow T1 \)
  - \( \text{Instr 2} \rightarrow T2 \)
  - \( \text{Instr 3} \rightarrow T3 \)
  - \( \text{Instr 4} \rightarrow T4 \)
  - \( \text{Instr 5} \rightarrow T5 \)
  - \( \text{Instr 6} \rightarrow T6 \)

- **Register File**
  - \( R1-R32 \)

- **Issue Queue (IQ)**
  - \( T1 \leftarrow R1+R2 \)
  - \( T2 \leftarrow T1+R3 \)
  - \( \text{BEQZ } T2 \)
  - \( T4 \leftarrow T1+T2 \)
  - \( T5 \leftarrow T4+T2 \)

- **ALU**
  - ALU
  - ALU
  - ALU

- **Results written to ROB and tags broadcast to IQ**
Title

• Bullet