Lecture 8: Speculation and Recovery

• Last Time
  - Multiple issue
  - Register renaming

• Today
  - Branch prediction
  - More sophisticated branch prediction algorithms
  - Branch target prediction
  - Speculation recovery

Slides courtesy of Stephen W. Keckler, UT-Austin

Branch Prediction

• Depending on use, some branches are very predictable
  - loops
    • TTT...TN
  - limit checks
    • almost always pass

• Some are not very predictable
  - data dependent dispatch with equally likely cases

• Types of predictors
  - static
  - history
  - multi-bit history
  - pattern

```c
for(j=0;j<30;j++) {
...
}
```

```c
switch(mode) {
case 1: ...
case 2: ...
default: ...
```

```c
if(a > limit) {
...
}
```
Static Prediction

- Assign a preferred direction to each branch
  - e.g.,
    - BNEZ_T (predict taken)
    - BNEZ_N (predict not taken)
- Base on
  - program analysis
    - loops tend to be taken
  - profiling of the program
    - but it may be data dependent

More static prediction

```c
for (initialize; condition; increment) {
    ...
}
```

![Diagram of loop control flow](attachment:loop_diagram.png)
Branch Performance

Consider a modern pipeline with a long decode stage

![Pipeline Diagram]

Discover its a branch and predict
Resolve direction and calculate target

Penalty for mispredicted branch:

If 20% of instructions are branches what is CPI
  With no prediction?
  How about predict not-taken (assume 50% accurate)
  With static branch prediction (assume 70% accurate)

Simple Dynamic Predictor (Local)

- Predict branch based on past history of branch
- Branch history table
  - indexed by PC (or fraction of it)
  - stores last direction each branch went
  - may indicate if last instruction at this address was a branch
  - table is a cache of recent branches
  - Buffer size of 4096 entries are common

- What happens if:
  - Don’t find PC in BHT?
  - Run out of BHT entries?

![Diagram of Simple Dynamic Predictor]
Performance of Dynamic Predictor

- Now we can predict it is a branch before decode
- Reduces delay to fetch next target
- Misprediction penalty is the same
- What is CPI assuming 85% prediction accuracy?

Multi-bit predictors

- A 'predict same as last' strategy gets two mispredicts on each loop
  - Predict NTTT...TTT
  - Actual TTTT...TTN
- Can do much better by adding inertia to the predictor
  - e.g., two-bit saturating counter
  - Predict TTTT...TTT
- Misprediction rates:
  - 4% (FP) to 11% (int)
### 2-bit counter

- 2-bits
- Increment on taken
- Decrement on not-taken
- MSB gives prediction

```
• N-bit counter
  if (counter > 2^(n-2)) {
    predict taken;
  } else {
    predict not-taken;
  }
```

• Compared to previous schemes, what is extra complexity?

### Summary of Basic Prediction

- Conventional architectures use branch prediction to solve the "fetch" problem
  - Direction Prediction - branch taken/not taken?
  - Target Prediction - address of next instruction?

- Branch predictors use the past history of branches to predict future branches
  - Current branch's direction can depend on the past history of this branch's behavior (local)
  - Current branch's direction can depend on the direction of the last \( n \) branches that were encountered before this branch (global)
    - Suppose pattern is TNNTNNTNN... how to predict this?
Predicting More Complex Behavior

- Pattern is TNNTNNTNN
- Correlated branches

```
if (d==0)
    d=1;
if (d==1)
    BNEZ R1,L1    ; b1 (d!=0)
    ADDI R1,R0,#1
    _L1: SUB R3,#1,R1
    BNEZ R3,L2    ; b2 (d!=1)
    _L2:
```

b1 and b2 are highly correlated!
- if b1 is not taken, then b2 is always taken

What happens when $d = 2,0,2,0,2...$
- 1 bit predictor
- 2 bit predictor

Global History Tables

- History gives a pattern of recent branches
  - e.g., TTNTNNTNN
  - what comes next?
- Predict next branch by looking up history of branches for a particular pattern
- Can combine PC and history register to distinguish between different branches
Per Branch Pattern History Table
(Two-Level Predictor)

- Two-level predictor
  - First level: find history (pattern)
  - 2nd level: predict branch for that pattern
- Correlating predictors
- Differs from global pattern history table as each branch has its own private history
- What structures must be updated?

Compaq Alpha 21264 Branch Predictor (1998)

- Local: previous executions of this branch
- Global: previous execution of all branches
- Tournament predictor
**Branch Prediction Accuracy**

- Static branch prediction (compiler) - 70%
  - Can get better with profiling (run program and examine branch directions)

- Per branch 2-bit saturating counters (no history) - 85%

- Two-level predictor (with history) - 90-95% accuracy

- Tournament predictor - a little more accurate than Two-level

- BUT - depends on benchmark
Current/Future Work in Prediction

• Make the predictor more accurate
  – Reduce effect of aliasing of different branches to same predictor entry
  – New algorithms
    • Machine learning (perceptrons, etc.)
    – When to update the predictor (speculatively or accurately)

• Reduce latency of predictor
  – HW/SW cooperative prediction

• Perform fewer predictions
  – Generate longer blocks of code without branches
    • Statically (block-oriented instruction sets)
    • Dynamically (trace caching)

Branch Target Tables

• Need to know where to go if the prediction is ‘taken’
  – predict the target along with the direction
  – JR instruction - target not known until after R stage!

• May use different target prediction strategy for different types of branches
  – subroutine returns

• Use a Branch Target Buffer (BTB) to predict target addresses
Branch Target Buffer

- Use current PC to index a cache of next PCs
  - Access simultaneously with branch predictor
  - If predictor says "taken" then use BTB result
  - Update BTB when result is computed
- Works well for branches
- What about JMP?
  - target is computed dynamically

Where Does the BTB fit in Pipeline?
Return Address Stack

- In general - computed jump targets can be hard to predict -
  BUT
- Virtual functions - not so bad
  - Use BTB
- Use a push-down stack to record subroutine return addresses
- The ISA can give hints about where you're going
- Digital Alpha has 4 instructions with identical ISA behavior
  - JMP, JSR, RET, JSR_COROUTINE
  - specify predictor's use of stack
  - include hint of target address
  - JMP R31, (R3), hint

Instruction Fetch Engine (Front-End)

- Keep the beast fed
  - Fetch multiple instructions per cycle
    - Easy if no branches (just make fetch path wider)
  - Branch prediction - fetch from predicted target
    - Do we allow predicted target instructions to be executed before instructions which we know must be executed? (Speculation)
    - Can't fetch from target until we predict the branch
    - May end up fetching multiple branches at once!
  - Predict the next memory (cache) line to fetch
    - Construct traces that have branch predictions built in (trace cache)
Summary

- **Branch prediction**
  - The past predicts the future (recognizing patterns)
  - Two-level predictors, hybrid predictors
  - Target prediction

- **Started discussion of rollback/recovery**

- **Next Time**
  - Finish commit logic and reorder/recovery
  - Handling memory dependencies