ILP: CONTROL FLOW

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Homework 2 submission deadline: Feb. 13th

- This lecture
  - Performance bottleneck
  - Program flow
  - Branch instructions
  - Branch prediction
Performance Bottleneck

- Key performance limitation
  - Number of instructions fetched per second is limited

- How to increase fetch performance?
  - Deeper fetch (multiple stages)
  - Wider fetch (multiple pipelines)

How to handle branches?
Impact of Branches

- Example C code
  - No structural/data hazards
  - What is fetch rate (IPS)?

- Five-stage pipeline
  - Cycle time = 10ns

Assembly code:
Loop: ADD R1, R1, R2
      ADDI R2, R2, #-1
      BNEQ R2, R0, Loop
      stall

d {  
    sum = sum + i;  
    i = i - 1;  
  } while(i > 0);
Impact of Branches

- Example C code
  - No structural/data hazards
  - What is fetch rate (IPS)?

- Ten-stage pipeline
  - Cycle time = 5ns

Assembly code:

```
do {
    sum = sum + i;
    i = i - 1;
} while(i > 0);
```

Loop:  
```
ADD R1, R1, R2
ADDI R2, R2, #-1
BNEQ R2, R0, Loop
stall
stall
stall
```

Fetch | Decode | Execute | Memory | Writeback
Program Flow

- A program contains basic blocks
  - Only one entry and one exit point per basic block

- Branches
  - Conditional vs. unconditional
    - How to check conditions
    - Jumps, calls, and returns
  - Target address
    - Absolute address
    - Relative to the program counter
Branch Instructions

- Branch penalty due to unknown outcome
  - Direction and target
- How to reduce penalty

Can we predict what instruction to be fetched?
Branch Prediction

- How to predict the outcome of a branch
  - Profiling the entire program
  - Predict based on common cases

Example C/C++ code:

```c
i = 10000;
do {
    r = i%4;
    if(r != 0) {
        sum = sum + i;
    }
    i = i - 1;
} while(i > 0);
```
Branch Prediction

- How to predict the outcome of a branch
  - Profiling the entire program
  - Predict based on common cases

Assembly code:

```
ADDI R1, R0, #10000

do:
  ANDI R2, R1, #3
  BEQ R2, R0, skp
  ADD R3, R3, R1

skp:
  ADDI R1, R1, #-1
  BNEQ R1, R0, do
```

<table>
<thead>
<tr>
<th>Branch</th>
<th>Taken</th>
<th>Not-Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch-1</td>
<td>2500</td>
<td>7500</td>
</tr>
<tr>
<td>branch-2</td>
<td>9999</td>
<td>1</td>
</tr>
</tbody>
</table>
Branch Prediction

- The goal of branch prediction
  - To avoid stall cycles in fetch stage

- Types
  - Static prediction (based on direction or profile)
    - Always not-taken
      - Target = next PC
    - Always taken
      - Target = unknown
  - Dynamic prediction
    - Special hardware using PC

Which ones are influenced
- a. Performance
- b. Energy
- c. Power
Branch Prediction/Misprediction

- Prediction accuracy?
  - A: always not-taken
    - 0.01
  - B: always taken
    - 0.99

```c
i = 100;
do {
    sum = sum + i;
    i = i - 1;
} while(i > 0);
```
Problem

- Compute IPC of a scalar processor when there are
  - no data/structural hazards, only control hazards,
  - every 5th instruction is a branch, and
  - 90% branch prediction accuracy

- \[ \text{IPC} = \frac{1}{1 + \text{stalls per instruction}} \]
  \[ = \frac{1}{1 + 0.2 \times 0.1 \times 1} = 0.98 \]
Dynamic Branch Prediction

- Hardware unit capable of learning at runtime
  - 1. Prediction logic
    - Direction (taken or not-taken)
    - Target address (where to fetch next)
  - 2. Outcome validation and training
    - Outcome is computed regardless of prediction
  - 3. Recovery from misprediction
    - Nullify the effect of instructions on the wrong path
Simple Dynamic Predictors

- One-bit branch predictor
  - Keep track of and use the outcome of last executed branch

- Prediction accuracy
  - A single predictor shared by multiple branches
  - Two mispredictions for loops (1 entry and 1 exit)

```c
while(1) {
    for(i=0; i<10; i++) {
    }
}
```