ILP: CONTROL FLOW

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

- Announcement
  - Homework 2 will be released on Sept. 26th

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

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

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

- How to increase fetch performance?
Performance Bottleneck

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

- How to increase fetch performance?
  - Wider fetch (multiple pipelines)
Performance Bottleneck

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

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

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

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

How to handle branches?
Impact of Branches

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

```c
do {
    sum = sum + i;
    i = i - 1;
} while(i != j);
```
Impact of Branches

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

- Five-stage pipeline
  - Cycle time = 10ns

```
do {
    sum = sum + i;
    i = i - 1;
} while(i != j);
```

Assembly code:
Loop:  ADD  R1, R1, R2
      ADDI  R2, R2, #1
      BNEQ R2, R0, Loop
      stall
Impact of Branches

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

- Ten-stage pipeline
  - Cycle time = 5ns

```c
do {
    sum = sum + i;
    i = i - 1;
} while(i != j);
```

Assembly code:

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

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>Execute</th>
<th>Memory</th>
<th>Writeback</th>
</tr>
</thead>
</table>

A program contains basic blocks

- Only one entry and one exit point per basic block
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
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
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);
```

How many branches?
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);
```

How many branches?
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
```
Branch Prediction

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

Assembly code:

<table>
<thead>
<tr>
<th>do:</th>
<th>TAKEN</th>
<th>NOT-TAKEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDI R1, R0, #10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>skp:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANDI R2, R1, #3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ R2, R0, skp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R3, R3, R1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>skp:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI R1, R1, #-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNEQ R1, R0, do</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

branch-1

branch-2
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></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
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
  - B: always taken

```c
i = 100;
do {
    sum = sum + i;
    i = i - 1;
} while(i != 0);
```
Branch Prediction/Misprediction

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

```plaintext
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
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

```c
while(1) {
    for(i=0; i<10; i++) {
    }
    for(j=0; j<20; j++) {
    }
}
```
Simple Dynamic Predictors

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

```plaintext
while:
    ADDI  R3, R0, #10
    JMP   chk1
for1: ...
chk1:  BNQ  R1, R3, for1
    ADDI  R3, R0, #20
    JMP   chk2
for2: ...
chk2:  BNQ  R2, R3, for2
    JMP   while
```

```plaintext
while(1) {
    for(i=0; i<10; i++) {
    }
    for(j=0; j<20; j++) {
    }
}
```

** Loop implementation suggested by Simon **
Simple Dynamic Predictors

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

- Prediction accuracy

```java
while(1) {
  for(i=0; i<10; i++) {
  }
  for(j=0; j<20; j++) {
  }
}
```
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)

```java
while(1) {
    for(i=0; i<10; i++) {
    }
    for(j=0; j<20; j++) {
    }
}
```