ILP: CONTROL FLOW

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

- Announcement
  - Homework 2 is due tonight (11:59PM)

- 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?**
Example C code

- No structural 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 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 hazards
  - What is fetch rate (IPS)?

- **Ten-stage pipeline**
  - Cycle time = 5ns

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

- Fetch | Decode | Execute | Memory | Writeback

- Ten-stage pipeline:
  - Cycle time = 5ns
  - Fetch | Decode | Execute | Memory | Writeback

- What is fetch rate (IPS)?
  - IPS: Instructions Per Second
A program contains basic blocks

Only one entry and one exit point per basic block
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
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:

```assembly
ADDI  R1, R0, #10000

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

skp:
  ADDI  R1, R1, #1-1
  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></td>
<td></td>
</tr>
<tr>
<td>branch-2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>Instruction</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>do:</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:
- Taken: 2500
- Not-Taken: 7500

Branch-2:
- Taken: 9999
- Not-Taken: 1
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
    - $0.99$

```
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

```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

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

**Loop implementation suggested by an student**
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

- 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++) {
  }
}
for(j=0; j<20; j++) {
}
```
Bimodal Branch Predictors

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

- Shared predictor

- Two mispredictions per loop

Accuracy = $26/30 = 0.86$

How to improve?
Bimodal Branch Predictors

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken

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

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken

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

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken

  - One misprediction on loop exit
  - Accuracy = $28/30 = 0.93$

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

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken

- One misprediction on loop exit
- Accuracy = 28/30 = 0.93

- How to improve?
  - 3-bit predictor?

- Problem?
  - A single predictor shared by many branches