PIPELINING: BRANCH AND MULTICYCLE INSTRUCTIONS

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

- Announcement
  - Homework 1 submission deadline: Jan. 30th

- This lecture
  - Control hazards in the five-stage pipeline
  - Multicycle instructions
    - Pipelined
    - Unpipelined
  - Reorder buffer
Control Hazards

- Example C/C++ code

```c
for (i=100; i > 0; i--) {
    sum = sum + i;
}
total = total + sum;
```

How many branches are in this code?
Control Hazards

- Example C/C++ code

```c
for (i=100; i > 0; i--)
{
    sum = sum + i;
}
total = total + sum;
```

What are possible target instructions?
Control Hazards

Example C/C++ code

```c
for (i=100; i > 0; i--) {
    sum = sum + i;
}

total = total + sum;
```

What happens inside the pipeline?
Handling Control Hazards

- 1. introducing stall cycles and delay slots

  - How many cycles/slots?
  - One branch per every six instructions on average!!

```plaintext
add r1, r0, #100
beq r0, r1, next
nothing
nothing
add r2, r2, r1
sub r1, r1, #1
J for
```

2 additional delay slots per 6 cycles!
Handling Control Hazards

1. introducing stall cycles and delay slots

- How many cycles/slots?
- One branch per every six instructions on average!!

```
add r1, r0, #100
for:  beq r0, r1, next
nothing
add r2, r2, r1
sub r1, r1, #1
J for
nothing
```

1 additional delay slot, but longer path
Handling Control Hazards

1. introducing stall cycles and delay slots
   - How many cycles/slots?
   - One branch per every six instructions on average!!

```
add r1, r0, #100
beq r0, r1, next
nothing
add r2, r2, r1
J for
sub r1, r1, #1
next: add r3, r3, r2
```
Handling Control Hazards

1. Introducing stall cycles and delay slots
   - How many cycles/slots?
   - One branch per every six instructions on average!

```
add r1, r0, #100
beq r0, r1, next
nothing
add r2, r2, r1
J for
sub r1, r1, #1
```

Jump and function calls can be resolved in the decode stage.
Handling Control Hazards

1. introducing stall cycles and delay slots
2. predict the branch outcome
   - simply assume the branch is taken or not taken
   - predict the next PC

```
add r1, r0, #100
beq r0, r1, next
add r2, r2, r1
sub r1, r1, #1
J for
add r3, r3, r2
```

May need to cancel the wrong path
Multicycle Instructions

- Not all of the ALU operations complete in one cycle
  - Typically, FP operations need more time
Multicycle Instructions

- Not all of the ALU operations complete in one cycle
  - pipelined and un-pipelined multicycle functional units

Pipelined vs. un-pipelined?
Multicycle Instructions

- Structural hazards
  - potentially multiple RF writes

Possibly multiple writes to the Register File
Multicycle Instructions

- Data hazards
  - more read-after-write hazards

<table>
<thead>
<tr>
<th>Load</th>
<th>Mul</th>
<th>Add</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>f4, 0(r2)</td>
<td>f0, f4, f6</td>
<td>f2, f0, f8</td>
<td>f2, 0(r2)</td>
</tr>
</tbody>
</table>

Instruction Flow:

- **IF**: Instruction Fetch
- **ID**: Instruction Decode
- **EX**: Execution
- **MA**: Memory Access
- **WB**: Write Back

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MA</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M3</td>
<td>M4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M5</td>
<td>M6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M7</td>
<td>MA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MA</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MA</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram shows the execution flow of the instructions with dependencies and hazards indicated.
Multicycle Instructions

- Data hazards
  - potential write-after-write hazards

```
load f4, 0(r2)
mul f2, f4, f6
add f2, f0, f8
store f2, 0(r2)
```

Out of Order Write-back!!
Multicycle Instructions

- Data hazards
  - potential write-after-write hazards

```
load f4, 0(r2)
mul f2, f4, f6
add f2, f0, f8
store f2, 0(r2)
```

In-Order Writes
Multicycle Instructions

- Imprecise exception
  - instructions do not necessarily complete in program order

```
load f4, 0(r2)  # IF ID EX MA WB
mul f2, f4, f6  # IF ID  M1 M2 M3 M4 M5 M6 M7 MA WB
add f3, f0, f8  # IF ID  A1 A2 A3 A4 MA WB
store f2, 0(r2) # IF ID  EX MA WB
```

Overflow!!
Multicycle Instructions

- Imprecise exception
  - state of the processor must be kept updated with respect to the program order

```
load f4, 0(r2)
mul f2, f4, f6
add f3, f0, f8
store f2, 0(r2)
```

In-order register file updates
Reorder Buffer

Multicycle Instructions

mul f2, f4, f6
add f4, f0, f1
sub f6, f3, f7
Reorder Buffer

- Multicycle Instructions

mul f2, f4, f6
add f4, f0, f1
sub f6, f3, f7