PIPELINING: BRANCH AND MULTICYCLE INSTRUCTIONS

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

- Announcement
  - Homework 2 release date: Sept. 11th

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

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Loading data from memory.

R1 ← Mem[R2]

R3 ← R1 + R0

R4 ← R1 - R3
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Loaded data will be available two cycles later.
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

**Inserting two bubbles.**

R1 ← Mem[R2]  
Nothing  
Nothing  
R3 ← R1+R0  
R4 ← R1-R3
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Inserting single bubble + RF bypassing.

R1 ← Mem[R2]

Nothing

R3 ← R1 + R0

R4 ← R1 - R3

Load delay slot.

SW vs. HW management?
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Using the result of an ALU instruction.

R1 ← R2+R3
R5 ← R1+R0
R3 ← R1+R0
R4 ← R1-R6
Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Using the result of an ALU instruction.

![Diagram showing data hazards with R1, R2, R3, R4, R5, R6, R1+R0, and R1+R0 with ALU result forwarding.]

Forwarding ALU result.
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)

Write must wait for earlier read

R1 ← R2 + R1
R2 ← R8 + R9
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
  - Write must wait for earlier read

R1 ← R2 + R1
R2 ← R8 + R9

No WAR hazards in 5-stage pipeline!
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- Output dependence: write-after-write (WAW)

[Old writes must not overwrite the younger write]

R1 ← R2 + R3

R1 ← R8 + R9
Data Hazards

- True dependence: read-after-write (RAW)
- Anti dependence: write-after-read (WAR)
- Output dependence: write-after-write (WAW)
  - Old writes must not overwrite the younger write

R1 $\leftarrow$ R2+R3

R1 $\leftarrow$ R8+R9

No WAW hazards in 5-stage pipeline!
Data Hazards

- Forwarding with additional hardware
How to detect and resolve data hazards

Show all of the data hazards in the code below

R1 ← Mem[R2]
R2 ← R1 + R0
R1 ← R1 - R2
Mem[R3] ← R2
Data Hazards

- How to detect and resolve data hazards
  - Show all of the data hazards in the code below

```
R1 ← Mem[R2]
R2 ← R1 + R0
R1 ← R1 - R2
Mem[R3] ← R2
```

- WAW
- WAR
- RAW
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!!

```
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!!

```
for:  add r1, r0, #100
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!!

<table>
<thead>
<tr>
<th>for:</th>
<th>add r1, r0, #100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>beq r0, r1, next</td>
</tr>
<tr>
<td></td>
<td>nothing</td>
</tr>
<tr>
<td></td>
<td>add r2, r2, r1</td>
</tr>
<tr>
<td></td>
<td>J for</td>
</tr>
<tr>
<td></td>
<td>sub r1, r1, #1</td>
</tr>
<tr>
<td></td>
<td>next: add r3, r3, r2</td>
</tr>
</tbody>
</table>

Reordering instructions may help
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
```

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
for: beq r0, r1, next
add r2, r2, r1
sub r1, r1, #1
J for
next: 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
Multicycle Instructions

- Data hazards
  - more read-after-write hazards

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

- Data hazards
  - more read-after-write hazards

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

- Data hazards
  - more read-after-write hazards

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

IF | ID | EX | MA | WB
---|----|----|----|----
M1 | M2 | M3 | M4 | M5 | M6 | M7 | MA | WB
A1 | A2 | A3 | A4 | MA | WB
ID | EX | MA | WB
```
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)
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)
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)
```

[Diagram showing the pipeline stages (IF, ID, EX, MA, WB) for the instructions]
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)
```
Multicycle Instructions

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

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

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

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

```
IF ID EX MA WB
M1 M2 M3 M4 M5 M6 M7 MA WB
ID A1 A2 A3 A4 MA WB
IF ID EX MA WB
```
Multicycle Instructions

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

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

```
Overflow!!
```
Imprecise exception

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MA</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>load f4, 0(r2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul f2, f4, f6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add f3, f0, f8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store f2, 0(r2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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