Complex Pipelines

Miguel Gomez-Garcia

(slides adapted from prior 6.823 offerings)
Dependence vs. hazard

• Dependence is a property of programs

• Whether a dependence results in a hazard is a property of pipeline organizations
Data hazard types

- **RAW**
- **WAR**
- **WAW**

---

I1: ADDI  f0, f0, 0
I2: ADDI  f3, f0, 3
I3: ADDI  f4, f0, 4
I4: ADDI  f0, f5, 1
I5: XOR   f6, f6, f6
I6: ADDI  f0, f7, 1

**Reads/Writes to f0**

- R (I1)
- W (I1)
- R (I2)
- R (I3)
- W (I4)
- W (I6)
Scoreboard

• A data structure that detects hazards dynamically

• Applicable to both in-order and out-of-order issue

• Why do we need this?
  • Many execution units
  • Variable execution latency
  • Dynamic instruction scheduling
Scoreboard

• Can have many implementations!
• Example: In-order issue
  • WAR cannot happen (if value is latched to functional unit at issue)
    I1: ADDI f1, f0, 1
    I2: ADDI f0, f2, 1
    Register read happens before write for an instruction
    Due to in-order issue

• Can be simplified as Busy[FU#] and WP[reg#] (if WAW resolved conservatively)
Scoreboard

• What strategy does it use to resolve RAW?
  • Stall

• How about bypass?
  • Less beneficial since the register write can happen right after execution finishes
  • Can still be incorporated to allow register read and write to happen in the same cycle
Out of Order (OoO) Execution

• Why use it in the first place?
  • Stalls of younger instructions prevent dispatch of younger instructions into functional (execution) units.

```
MUL  R3 <- R1, R2
ADD  R3 <- R3, R1
ADD  R1 <- R6, R7
MUL  R5 <- R6, R8
ADD  R7 <- R3, R5

LD   R3 <- R1 (0)
ADD  R3 <- R3, R1
ADD  R1 <- R6, R7
MUL  R5 <- R6, R8
ADD  R7 <- R3, R5
```

• By eliding false dependences and head-of-line blocking, we are only bound by true data dependences
OoO execution dynamically extracts true dependences

Decode

Commit

Not ready
Ready
Complete
Committed

Execute

(Not decoded)
Out-of-order execution

• Want: we want to somehow avoid stalling due to WAR and WAW hazards...
  • Strategy?
    Do something else

• Technique?
  Register Renaming
Renaming + ROB

Renaming table & reg file

<table>
<thead>
<tr>
<th>p</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>v1</td>
</tr>
<tr>
<td>F3</td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>t8</td>
</tr>
<tr>
<td>F5</td>
<td></td>
</tr>
<tr>
<td>F6</td>
<td>t3</td>
</tr>
<tr>
<td>F7</td>
<td></td>
</tr>
<tr>
<td>F8</td>
<td>v4</td>
</tr>
</tbody>
</table>

data (v_i) / tag(t_i)

Reorder buffer

<table>
<thead>
<tr>
<th>Ins#</th>
<th>use</th>
<th>exec</th>
<th>op</th>
<th>p1</th>
<th>src1</th>
<th>p2</th>
<th>src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>LD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>LD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>MUL</td>
<td>0</td>
<td>v2</td>
<td>1</td>
<td>v1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>SUB</td>
<td>1</td>
<td>v1</td>
<td>1</td>
<td>v1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>DIV</td>
<td>1</td>
<td>v1</td>
<td>0</td>
<td>v4</td>
</tr>
</tbody>
</table>

- Insert instruction in ROB
- Issue instruction from ROB
- Complete instruction
- Empty ROB entry

What happens if I3 raises exception?
Precise Exceptions

• Issue:
  • Exception ordering is not accurate
  • Instructions that come earlier in program order might not raise an exception until after later instructions have been completed!

• Solution:
  • Introduce in-order commit point!

But where do we store data before commit?
Exception-friendly ROB

Register File
(now holds only committed state)

Reorder buffer

Search dest field for renaming info

Load Unit
FU
FU
FU
Store Unit
Commit

< t, result >
Renaming Table

Register File

Reorder buffer

Rename Table

<table>
<thead>
<tr>
<th>Ins#</th>
<th>use</th>
<th>exec</th>
<th>op</th>
<th>p1</th>
<th>src1</th>
<th>b2</th>
<th>src2</th>
<th>pd</th>
<th>dest</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$t_1$

$t_2$

$\ldots$

$t_n$

$< t, \text{result}>$

Load Unit

FU

FU

FU

Store Unit

Commit
Questions?